THE HONG KONG UNIVERSIT OF SCIENCE AND TECHNOLOGY | LIBRARY

HKUST SPD - INSTITUTIONAL REPOSITORY

Title	Reinforcement Learning Based Query Vertex Ordering Model for Subgraph Matching
Authors	Wang, Hanchen; Zhang, Ying; Qin, Lu; Wang, Wei; Zhang, Wenjie; Lin, Xuemin
Source	Proceedings: 2022 IEEE 38th International Conference on Data Engineering, v. 2022, May 2022
Conference	2022 38th IEEE International Conference on Data Engineering, ICDE 2022, Virtual; Kuala Lumpur, Malaysia, 9 - 12 May 2022
Version	Accepted Version
DOI	10.1109/icde53745.2022.00023
Publisher	IEEE
Copyright	© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This version is available at HKUST SPD - Institutional Repository (https://repository.hkust.edu.hk/ir)

If it is the author's pre-published version, changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published version.

Reinforcement Learning Based Query Vertex Ordering Model for Subgraph Matching

Hanchen Wang[♭‡], Ying Zhang[♭†*], Lu Qin[†], Wei Wang[♮], Wenjie Zhang[‡], Xuemin Lin[‡]
[♭]Zhejiang Gongshang University,[‡]University of New South Wales,
[†]University of Technology Sydney, [♮]Hong Kong University of Science and Technology
{hanchen.wang, ying.zhang, lu.qin}@uts.edu.au, weiwcs@ust.hk, {zhangw, lxue}@cse.unsw.edu.au

***Abstract*—Subgraph matching is a fundamental problem in various fields that use graph structured data. Subgraph matching algorithms enumerate all isomorphic embeddings of a query graph q in a data graph G. An important branch of matching algorithms exploit the backtracking search approach which recursively extends intermediate results following a matching order of query vertices. It has been shown that the matching order plays a critical role in time efficiency of these backtracking based subgraph matching algorithms. In recent years, many advanced techniques for query vertex ordering (i.e., matching order generation) have been proposed to reduce the unpromising intermediate results according to the preset heuristic rules. In this paper, for the first time we apply the Reinforcement Learning (RL) and Graph Neural Networks (GNNs) techniques to generate the high-quality matching order for subgraph matching algorithms. Instead of using the fixed heuristics to generate the matching order, our model could capture and make full use of the graph information, and thus determine the query vertex order with the adaptive learning-based rule that could significantly reduces the number of redundant enumerations. With the help of the reinforcement learning framework, our model is able to consider the long-term benefits rather than only consider the local information at current ordering step. Extensive experiments on six real-life data graphs demonstrate that our proposed matching order generation technique could reduce up to two orders of magnitude of query processing time compared to the state-of-the-art algorithms.**

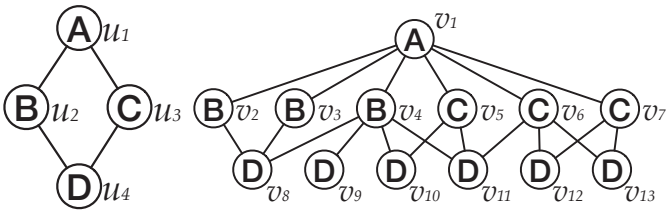
***Index Terms*—Subgraph Matching, Reinforcement Learning, Graph Neural Network**

I. Introduction

In recent years, graph structured data has been increasingly ubiquitous. Graph analysis also becomes much more important and popular in the area of data analytics. Subgraph matching is one of the most fundamental problems in graph analysis. Subgraph matching aims to find all embeddings in a data graph G that are isomorphic to a query graph q. For example, in Figure 1, $\{(u_1, v_1), (u_2, v_4), (u_3, v_5), (u_4, v_{10})\}$ is a match from query graph q to data graph G. Due to its importance, subgraph matching is widely used in various areas in both academia and industry [1]–[4].

Subgraph matching has been proved to be a NP-complete problem [5], which means that the computational complexity is factorial in the worst case. Though this limit is intrinsic and cannot be overcome, great research efforts still lead to significant advances in reducing unpromising intermediate results, and thus reduce the computational complexity in the average case.

The subgraph matching algorithms can be categorized in two major classes by their main methodologies: join-based algorithms and backtracking search based algorithms. The join-based approaches [6]–[8] convert the query q to a multi-way join by mapping vertices and edges in q to attributes and relations. These approaches evaluate the multi-way join to find matching results. Another class of approaches are the backtracking search based algorithms, which recursively extend intermediate results by mapping query vertices to data vertices along an order of query vertices [9]. Many methods adopt the state space representation for subgraph matching, where each state represents an intermediate result [10], *i.e.,* a mapping from query vertex to data vertex. The methods explore the state space with given order to find feasible state as a match. Among these methods, the query vertex order plays an important role in reducing the average time complexity for subgraph matching, because a well-designed query vertex order could enable the algorithm to prune out the unpromising results as much as possible [11]–[13].



(a) Query Graph q (b) Data Graph G

Fig. 1: Example query graph and data graph.

In this paper, we focus on the backtracking search based subgraph matching algorithms. As shown in [14], it is critical to choose a good matching order for the backtracking search based algorithms because it will greatly affect the search performance. Therefore, existing backtracking based algorithms put a lot of efforts to generate good matching orders. Clearly, finding the optimal matching order is time-prohibitive, and existing algorithms resort to a variety of heuristic strategies. For instance, QuickSI [15] proposed a *infrequent-edge first* ordering method which converts q into a weighted graph based on the label frequency of query edges in data graph and generates the matching order along the ascending order of their weights. RI [16] generates the order based on the structure of q, which selects the vertex with the maximum degree in q as the first vertices, and then picks the vertex that has the most neighbors in order ϕ as the next node. Other methods such as VF2++ [17], GraphQL [18], CFL [11] CECI [19], DP-iso [12] and VEQ [20] generate the order ϕ by building auxiliary structure of q based on the degrees, labels and candidate sizes of query vertices.

*Corresponding Author

However, these ordering methods only utilize the local structural and label information by greedy heuristics. Consequently, these methods are lacking the ability of fully exploiting the information within the query and data graphs; meanwhile, these methods can only obtain the locally optimal order produced by the greedy search algorithm. The auxiliary structure-based query ordering methods in CFL and DP-iso, which dynamically generate matching orders, can result in a large number of unsolved queries (*i.e.,* whose query processing time exceeds the preset time limit), and hence cannot achieve the state-of-the-art search performance according to the empirical study [14].

Motivations and Our Approaches

In this paper, we propose the first Reinforcement Learning based Query Vertex Ordering method for backtracking based subgraph matching, namely RL-QVO, to fill this research gap. RL-QVO aims to be a more efficient and scalable technique to conduct subgraph matching. We introduce the motivations in this paper as follows:

Motivation 1: Make Full Use of the Graph Information. As mentioned previously, the recent subgraph matching models generate ϕ based on the preset priority of heuristics such as degree, label frequency, path frequency, candidate size and etc. In practice, the order is usually determined by partial heuristics, and other statistics are somehow ignored. Besides, the priority of the heuristics cannot be adapted for graphs with different distributions. Consider q and G in Figure 1, if we use the ordering method of RI which selects the first node with the highest degree, the order will be generated randomly due to the structural symmetry of q. Accordingly, the following order is also generated on a random basis. Clearly, the order generated by RI cannot guarantee to reduce the redundant intermediate results for this query. Otherwise, if we build the order according to the label frequency in this case, vertex v_1 will be selected as the starting vertex in ϕ, which eventually reduces unpromising intermediate results. Therefore, adaptive priority for heuristics under particular query graph could significantly enhance the ordering methods, which requires the whole picture of graph information and decision making based on these information.

Our Approach: Capture the Graph Information with Graph Neural Network. Regarding the above observation, we exploit the graph neural networks [21]–[23] whose ability of feature extraction and representation learning has been widely proven theoretically and empirically, to capture the information hidden in q. We carefully design the initial feature for each query vertex based on the popular heuristics. With the help of GNN's message passing for neighbor aggregation, RL-QVO can naturally capture the query graph's structural and attribute information. Furthermore, thanks to the generality of the machine learning based techniques, RL-QVO is able to select the next node to generate the matching order for different query graphs according to learned vertex representations.

Motivation 2: Limitation of Greedy Heuristics. Most existing ordering methods utilize the greedy search methods with preset rules which will intrinsically fall into local optimum. Such methods intend to add the vertex that could reduce the most redundant intermediate results to the order ϕ at every step. However, these methods cannot consider the long-term query time cost (*i.e.,* quality of the order). The exact optimal order can only be found after all possible order permutations are evaluated. Therefore, it is challenging for us to develop a subgraph matching algorithm that can overcome the limitation of the greedy criterion.

Our Approach: Overcoming the Limitation with Reinforcement Learning. Motivated by this drawback, we design a reinforcement learning based framework to learn a policy for matching order generation. We craft the long-term cumulative reward for the RL-based model to force the model to consider the overall computational cost when selecting the next node for matching order ϕ. Instead of only focusing on the current step, the RL-based framework allows the learned policy to consider the states multiple steps ahead before making the decision. We also design an entropy reward to encourage the model to explore the unvisited states. Instead of enumerating all possible orders and evaluating the quality of each of them, RL-QVO samples a subset of such orders and learns the policy network from the observed rewards. Accordingly, RL-QVO has higher probability to produce matching orders that are closer to the optimal ones compared to the existing ordering methods with acceptable overhead.

Contributions

The contributions of our work are summarized as follows:

- To the best of our knowledge, our proposed RL-QVO is the *first* work to employ Reinforcement Learning technique to obtain high-quality matching order for backtracking based subgraph matching algorithms.
- Our RL-based model could fully exploit the graph information and select the next query node for the matching order at every step according to the cumulative reward, which can consider the long-term benefits.
- Extensive experiments conducted on 6 real-life graphs demonstrate that the high-quality matching order obtained by RL-QVO can significantly save the query processing time by up to two orders of magnitude.

Organization. The rest of the paper is organized as follows: Section II includes the preliminaries, definitions and related works. Section III presents the details of our proposed model RL-QVO. We report the experiment results in Section IV and give a conclusion in Section V.

II. Background and Related Work

In this Section, we introduce the preliminaries, problem statement, state-of-the-art subgraph matching algorithms and related works.

A. Preliminaries

In this paper, we focus on the subgraph matching problem on the undirected labeled graph $G = (V, E)$, where V denotes the vertices set, E is a set of edges. There is a label function f_l which maps a vertex $v \in V$ into a label l in the label set L. The query graph, denoted by q, is a potential subgraph of data graph G. q and G share the same label function f_l and the same label set L. The frequently used notations are summarized in the Table I.

We give the key formal definition of the subgraph isomorphism as follows:

TABLE I: The summary of notations

Notation	Definition
g, q, G	graph, query graph and data graph.
V, E, L	vertex set, edge set and label set.
f_l, f_{iso}	label mapping function and subgraph isomorphism function.
$e(u,v)$, $N(v)$	an edge between node u, v and the neighbor set of v.
$d(u)$	degree of node u.
C, LC	set of candidate vertices and local candidate vertices.
ϕ, $\#_{enum}$	matching order and enumeration number.
$N_+^{\phi}(u), N_-^{\phi}(u)$	backward and forward neighbors of u given ϕ.

Algorithm 1: Generic Subgraph Matching

Input: Query graph q, data graph G.
Output: Subgraph Match Mapping M from q to G.
// The filtering method.
1 $C \leftarrow$ generate candidate vertex sets
// The ordering method.
2 $\phi \leftarrow$ generate a matching order based on q, G and C
// The enumeration procedure, finds the mapping M for all query nodes in q.
3 $M = \texttt{Enumerate}(q, G, C, \phi, \{\}, 1)$

Definition II.1 (Subgraph Isomorphism)**.** Given a query graph $q = (V, E)$ and a data graph $G = (V', E')$, a subgraph isomorphism is an injective function f_{iso} from V to V' such that (1) $\forall v \in V, f_l(v) = f_l(f_{iso}(v))$; and (2) $\forall e_{(u,v)} \in E, e_{(f_{iso}(u), f_{iso}(v))} \in E'$.

Subgraph Matching. The objective of the subgraph matching is searching for all subgraph isomorphisms from query graph q to data graph G. Generally, the existing subgraph matching algorithms can be classified into two categories: backtracking based methods and join-based methods. Both directions have been intensively studied in the literature. In this paper, we focus on the backtracking based subgraph matching methods.

As summarized in previous works [14], [24], backtracking based subgraph matching method can be partitioned into three phases: (1) the *complete candidate vertex set* generation; (2) matching *order* generation; and (3) *matching enumeration*. This general framework (from [14]) is outlined in Algorithm 1.

In more details, **Phase (1)** is to pre-process the graph where various filtering techniques are deployed to generate candidate vertex sets C for all query nodes; this can reduce the search space before the enumeration process begins.

Definition II.2 (Complete Candidate Vertex Set $\mathcal{C}$)**.** Given q and G, a complete candidate vertex set $C(u)$ of $u \in V(q)$ is a set of data vertices such that for each $v \in V(G)$, if (u, v) exists in a match from q to G, then $v \in C(u)$.

In **Phase (2)**, a match order ϕ is generated to guide the enumeration of the matched subgraphs, which is formally defined as follows.

Definition II.3 (Matching Order)**.** A matching order ϕ is a permutation (i.e., sequence) of query graph's vertex set $V(q)$.

Definition II.4 (Backward (Forward) Neighbors)**.** With given order ϕ, the backward (forward) neighbors $N_+^{\phi}(u)$ ($N_-^{\phi}(u)$) are the neighbors of u positioned before (after) u in ϕ.

Usually, the matching order is generated heuristically based on the label frequency, degree of vertices and other structural and attribute information. For example, QSI introduces an *infrequent-edge first ordering method*, VF2++ uses a *infrequent-label first order* and CFL proposes a *path-based ordering method*.

Following is the formal definition of enumeration procedure in **Phase (3)**, which finds all matches of the query subgraph q in the data graph G.

Definition II.5 (Enumeration Procedure)**.** An enumeration procedure is performed recursively to find subgraph matches f_{iso} with given matching order ϕ and candidate vertex set C.

B. Problem Statement

As highlighted in [14], it is critical to choose a good matching order since the enumeration number in the search is significantly influenced by the matching order, which is strongly correlated to the query time. Following is a formal definition of the enumeration number.

Definition II.6 (Enumeration Number)**.** An enumeration number $\#_{enum}$ is the number of *recursive* calls of the enumeration procedure to find all matches with given q, G, ϕ and C.

It is cost-prohibitive to find an *optimal* matching order, and existing algorithms resort to heuristic strategies to find a good matching order. Inspired by the great successes of RL techniques in a variety of optimization problems [25], in this paper, we **aim to apply Reinforcement Learning technique to find a matching order** for given query graph q and data graph G to **minimize the enumeration number** in the matching enumeration phase such that the search performance of the subgraph matching can be enhanced.

C. State-of-the-Art

In this subsection, we introduce the backtracking search based subgraph matching algorithm Hybrid which is shown to achieve state-of-the-art search performance according to the intensive empirical results [14], and the recent published algorithm VEQ [20].

Hybrid utilizes the candidate filtering, vertex ordering and enumeration methods of GraphQL [18], RI [16] and QuickSI [15] respectively, which are introduced as follows:

(1) Candidate Set Generation. To limit the size of candidate vertex set for each node in the query, Hybrid utilizes the candidate vertex filtering method in GraphQL [18]. Particularly, Hybrid generates the candidate set with local pruning and global refinement. Local pruning filters out the invalid nodes based on the *profile* of the neighborhood graph of the query vertex u, which is the lexicographic order labels of u and neighbors of u. If the profile of query vertex u is a sub-sequence of that of data vertex v, then v is added to $C(u)$. Global refinement prunes $C(u)$ as follows: given $v \in C(u)$, the algorithm first build a bipartite graph B_u^v between $N(u)$ and $N(v)$ by adding $e(u', v')$ where $u' \in N(u)$ and $v' \in N(v)$, if $u' \in C(v')$. Global refinement then checks whether there is a semi-perfect matching in B_v^u, *i.e.,* whether all vertices in $N(v)$ are matched. If not, v will be removed from $C(u)$.

(2) Matching Order Generation. Query vertex order generation is one of the key factors that reduce the query processing time. Hybrid exploits RI's order generation method which is the state-of-the-art ordering method [14]. Hybrid generates the matching order only based on the structure of query graph

Algorithm 2: Enumeration Procedure

Input: Candidate set C, query graph q, data graph G, match order ϕ.
Output: Subgraph Match Mapping M.

```
1  Enumerate(q, G, C, φ, M, i)
   // Start with Enumerate(q, G, C, φ, {}, 1)
2  begin
3      if i > |φ| then
4          Output M, return.
5      u ← select an extendable vertex given φ and M
       // Compute local candidate set
6      LC(u, M) ← ComputeLC(q, G, C, φ, M, u, i)
7      for v ∈ LC(u, M) do
8          if v ∉ M then
9              Add (u, v) to M
10             Enumerate(q, G, C, φ, M, i + 1)
11             Remove (u, v) from M
```

q. Hybrid first selects the query node with the maximum degree $u* = \arg\max_{u \in V(q)} d(u)$ as the starting nodes of order ϕ. We denote the order determined at the t-th step as ϕ_t. Then the nodes with the most neighbors in ϕ_t are iteratively selected, *i.e.,* $u* = \arg\max_{u \in V(q) \wedge u \notin \phi_t} |N(u) \cap \phi_t|$. Since ties can easily occur in the $\arg\max$ function, Hybrid breaks the ties by considering following properties in order of: (1) the maximum value of $|\{u' \in \phi_t \mid \exists u'' \in V(q) \setminus \phi_t, e(u', u'') \in E(q) \wedge e(u, u'') \in E(q)\}|$, *i.e.,* the number of vertices in ϕ_t that have a neighbor outside of ϕ_t and is adjacent with u. The number is denoted as $|u_{neig}|$. (2) the maximum number of neighbors of u that are not in ϕ, and not even adjacent with vertices in ϕ_t, which is defined as $|u_{unv}| = |\{u' \in N(u) - \phi_t | \forall u'' \in \phi, e(u', u'') \notin E(q)\}|$. If these values are still the same for at least two nodes, Hybrid will choose a node arbitrarily.

Since Hybrid only considers the structure of query graph to build the matching order, it ignores the abundant information of the relationship between the query and data graphs. Consequently, Hybrid has to apply random selections in many cases, even selects the starting vertex arbitrarily. This selection cannot guarantee to produce a robust matching order to reduce redundant intermediate results.

(3) Enumeration Procedure. Hybrid adopts the recursive enumeration procedure which is also widely used in existing backtracking search based methods. The enumeration procedure is summarized in Algorithm 2. The function `Enumerate(·)` recursively enumerates all results. M is a set that records all mappings from query vertices to the data graph vertices. Once all the vertices of the query graph are mapped, Line 4 outputs the mapping M. Otherwise, the algorithm selects an extendable vertex which is defined as follows:

Definition II.7 (Extendable Vertices)**.** Given matching order ϕ and mapping M, extendable vertices $\Gamma(\phi, M)$ are query vertices v such that each $u' \in N_+^{\phi}(u)$ has been mapped in M but u has not.

Line 6 shows the computation of the local candidate set after selecting the extendable vertex. The local candidate set is computed following different rules for specific subgraph matching algorithms. Hybrid generates the local candidate set $LC(\cdot)$ by checking whether the vertices in candidate set are connected to the last matched data vertices. Line 7-11 loop over $LC(u, M)$ to find more mappings to extend M, and recursively invoke the `Enumerate(·)` function. Finally, we get the matching from $q(\phi(1:i))$ to G.

VEQ [20] is a recent published algorithm for subgraph matching problem. VEQ follows the three-phase framework mentioned above.

(1) Candidate Set Generation. VEQ first builds the query directed acyclic graph (DAG) of query graph q by assigning directions to edges in q. VEQ also finds the neighbor equivalence class (NEC) for degree-one vertices, in which the query vertices have the same label and the same neighbors. VEQ then builds the candidate set by using extended DAG-graph dynamic programming with an additional filtering function that utilizes a concept called neighbor-safty.

(2) Matching Order Generation. VEQ generates the matching order based on the size of candidate set and the size of neighbor equivalence class (NEC), which could reduce the redundancy in search space. However, this order generation method is still based on greedy heuristics and might be trapped in local optimum.

(3) Enumeration Procedure. VEQ follows the same enumeration procedure introduced above. Besides, VEQ prunes out repetitive subtrees of the search space by utilizing dynamic equivalence of the subtrees.

In this paper, we design the RL-based matching order generation method to improve the overall search performance. Specifically, we utilize the same candidate set generation and enumeration methods as Hybrid.

D. Related Work

In this subsection, we introduce the works which are closely related to your study.

Subgraph Matching. There are two major categories of subgraph matching methods. There are great number of subgraph matching models can be categorized as backtracking search based algorithms [10]–[13], [15], [17], [20], [26], [27]. These models follow the filtering, ordering and enumeration framework to obtain matches for query graphs. The other category is the join-based methods [6], [7], [28] which usually aim at the subgraph enumeration problem. These algorithms convert subgraph enumeration problem to a multi-way join task, which are also utilized in many database systems [29], [30]. In this paper, we aim at designing an ordering method for backtracking search based algorithms. Besides, there is a family of subgraph matching methods [31]–[33] which are based on constrained programming. These methods are proven proven optimal for several mid-sized or highly complicated cases. There are also several algorithms [34]–[36] utilize indexing-enumeration framework, which constructs indices on G and answers all queries with the assistance of the indices. These index-based methods would outperform the non-index-based methods in many cases, please refer to [37] for detailed experimental survey. However, these algorithms may have severe scalability issues according to [14], [37], [38].

Recently, there have been several research attempts [39]–[41] to develop subgraph matching algorithms on a learning

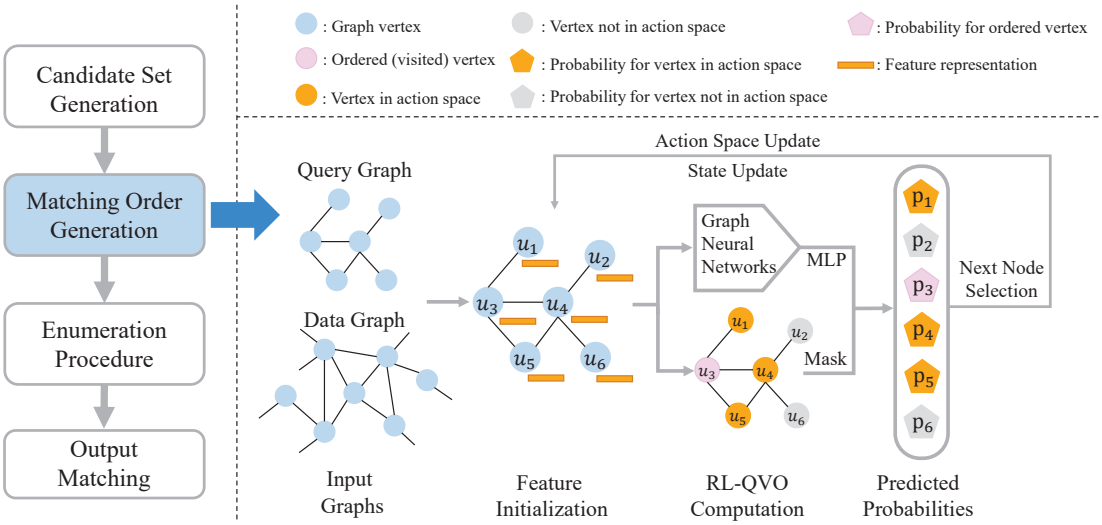


Fig. 2: Framework of RL-QVO

basis. These methods focus on counting of isomorphic subgraphs or producing approximate matching results, which are different from the target in this paper, *i.e.,* generating matching orders to find exact subgraph mappings.

Reinforcement Learning. Using reinforcement learning for graph-related problems is becoming increasingly popular. [42]–[46] aim to solve the NP-hard combinatorial optimization problems with reinforcement learning and graph representation learning techniques. Please see [47] for detailed survey. Several RL-based models have been proposed for graph analysis tasks. For example, [48] exploits a GNN-based policy network in promoting the performance for cross-domain graph analysis tasks such as classification; [49], [50] utilize graph policy networks on the molecular graphs for chemical and medical applications such as chemical reaction prediction.

Graph Neural Networks. Graph neural networks (GNNs) have shown great success in various graph-related applications [51]–[56]. Typically, GNNs learn the node representations with iterative aggregation of neighborhood information for each node, e.g., GCN [21], GAT [22] and GraphSAGE [57]. Despite these models, there have been a great number of GNNs proposed for wide range of real-life applications, please refer to [23] for comprehensive survey.

III. Our Approach

In this section, we present a reinforcement learning based model RL-QVO to generate high-quality matching orders.

A. Motivation

The backtracking-based subgraph matching methods explore the search space with a given matching order, which presents several unique challenges that are good fits for reinforcement learning and graph neural network:

First, it is cost-prohibitive to find the optimal matching order for a given query graph q and a data graph G. The existing ordering methods adopt the greedy heuristics to pick nodes that could possibly reduce the search space at current step as early as possible. However, these selections may result in the global *suboptimality*. With the help of reinforcement learning technique, our model is able to generate the matching order by considering the long-term rewards multi-steps ahead. Therefore, our proposed RL-QVO has better chance to escape from local optimum, and eventually shows higher matching order quality than the greedy methods.

Second, existing ordering methods only utilize the local neighborhood information with fixed priority to produce the matching order, which has a non-negligible probability to ignore some structural and attribute information within q and G. Graph neural networks (GNNs) have already been proven to have powerful ability of exacting information for specific graph analysis tasks. With the reinforcement learning framework, RL-QVO does not need to make assumptions on the query or data graph distributions, but regards the ordering process as a black-box computation on representations learned by GNNs with carefully designed initial features. As a result, RL-QVO generates the matching order adaptively with regards to different query graphs.

These motivate us to apply the Graph Neural Networks and Reinforcement Learning techniques to better utilize the graph information and find high-quality matching orders by looking multi-steps ahead (i.e., long-term rewards) during the training process. Though the training of the model needs extra overhead compared to existing matching order generation approaches, it can be preprocessed which is a common practice for various indexing techniques in the literature. Moreover, we show that the generation of the matching order during the query processing is very time efficient, and the gain is significant as demonstrated in our experiments.

There are three major categories of reinforcement learning algorithms: value function based methods, policy search methods and the methods combine both value function and policy search such as actor-critics [58].

We observed in our initial experiments that the enumeration numbers for the query vary vastly with different matching orders. Therefore, the methods use value function, such as Q-learning and actor-critics, are hard to converge, which leads to unsatisfactory results. According to these observations, we choose to apply policy search method in our model.

B. Framework

The research focus of this paper is to design a novel model to generate good matching orders for backtracking based subgraph matching algorithms. Thus, we replace the matching order generation part of the the state-of-the-art algorithm Hybrid with a new model: **Reinforcement Learning Based Query Vertex Ordering Model** (**RL-QVO** for short).

Figure 2 illustrates the framework of the RL-QVO. Particularly, RL-QVO regards the matching order generation as a Markov Decision Process (MDP), and the vertices in a matching order are generated sequentially. At the beginning, RL-QVO first aggregates neighbor information with carefully designed features to produce node representations for each query vertex. With the obtained representations, RL-QVO computes the probability scores for query vertices to guide the selection of the next query node for the matching order. The matching order is output if all query nodes are processed. Otherwise, we will repeat this procedure with updated action space and features. More details of the model will be introduced in the following subsections.

C. Query Vertex Ordering as Markov Decision Process

The query vertex ordering process can be naturally formulated as a Markov Decision Process (MDP). Given ordered nodes ϕ_t and input feature matrix $\boldsymbol{H}^t$ of query graph at step t as a state, reinforcement learning method takes an action from action space determined by the neighbors of the ordered vertices $N(\phi_t) = \{N(u)|\forall u \in \phi_t, N(u) \notin \phi_t\}$ with predicted probabilities to select next node for the order ϕ_{t+1}. After every selection, the feature matrix $\boldsymbol{H}$ and action space are also updated. With the generated order ϕ, the model performs the enumeration procedure and gets the reward based on the reduced number of enumerations $\Delta\#_{enum}$ compared to baselines, entropy of predicted probabilities and number of valid predictions. The MDP is formally defined as follows:

State. At step t, the state is defined by the order ϕ_t which contains t vertices and query graph representation matrix $\boldsymbol{H}_q^t$. The whole query graph feature matrix $\boldsymbol{H}_q$ is involve in the state representation in order to familiarize the model with the overall structural and attributed information of the query graph. Since RL-QVO is a machine learning-based model, for each vertex in query graph, we carefully initialize a feature representation $\boldsymbol{h}_u^{(0)}$.

Feature Representations. As discussed in Section II-A, the statistical heuristics of query graphs, such as degree of query vertices, label frequency and *etc*, play a key role in determining the query vertex order. Inspired by [48], we initialize the input features for query vertices based on important heuristics in order to fully exploit the information and relations hidden in and between query and data graphs. Specifically, we generate the initial feature as follows:

We compute the degree of each node. Intuitively, the query node with greater degree will have less matching in the data graph, and eventually has higher priority to be added in the order ϕ. Therefore, we use a scaled degree as one of the initial state value, *i.e.,*

$$\boldsymbol{h}_u^{(0)}(1) = degree(u)/\alpha_{degree},$$

where α_{degree} is a scaling factor to ensure the computation stability.

The initial feature also includes the node label information. Since integers are used to represent labels in our data format, we simply put the integer label id into the representation for each query vertex.

$$\boldsymbol{h}_u^{(0)}(2) = label(u)$$

To enable the graph neural network used in RL-QVO to discriminate the order of input node, we directly put the query node id in the initial representation.

$$\boldsymbol{h}_u^{(0)}(3) = id(u)$$

Please note that the query graph usually has small number of nodes, therefore there is no need to perform scaling on the vertex id.

We further include the data graph related heuristics to build our initial feature representation. The following statistics are commonly used in existing models: the frequency of vertices v in the data graph G with greater degree than query vertex u; and the frequency of vertices v in the data graph that have the same label as query vertex u. They are vectorized in the initial feature representation.

$$\boldsymbol{h}_u^{(0)}(4) = |\{v \in G|d(u) < d(v)\}| \,/(|V(G)| \times \alpha_d);$$

$$\boldsymbol{h}_u^{(0)}(5) = |\{v \in G|L(u) = L(v)\}| \,/(|V(G)| \times \alpha_l);$$

where α_d and α_l are the scaling factors. Please note that the frequency computations of $\boldsymbol{h}_u^{(0)}(4)$ and $\boldsymbol{h}_u^{(0)}(5)$ involve all data vertices, which reflect the distribution of the data graph.

Lastly, we put the number of unordered vertices and a trailing indicator at the initial representation. The number of unordered vertices enables the model to make different decision at different time step t, which is formulated as:

$$\boldsymbol{h}_u^t(6) = |V(q)| - t + 1$$

The indicator variable is 0 if the vertex u has not been ordered before the selection at time step t (*i.e.,* u is not in the order ϕ_{t-1}) and 1 otherwise, *i.e.,*

$$\boldsymbol{h}_u^t(7) = \mathbb{1}(u \in \phi_{t-1})$$

Before the selection starts, *i.e.,* when $t = 0$, indicators are 0 for all query vertices, and the indicator variable will be updated with the change of the state at every time step.

We concatenate all these features to formulate the input representation. This representation could easily incorporate additional heuristic features by appending these features to our initial feature representation vector. In our model, we only use five primary heuristics introduced above with two trailing indicators to capture the differences between the query vertices, and exploit the reinforcement learning and graph neural networks to automatically learn more complicated and informative criterion for query vertex order generation.

Action. The action space is defined by the neighbor vertices set $N(\phi_t) = \{N(u)|\forall u \in \phi_t, N(u) \notin \phi_t\}$ of the ordered vertices at current step t. At every step t, the action is to select vertex u' from $N(\phi_t)$ according to the predicted probabilities and add u' into the matching order for next step, *i.e.,* $\phi_{t+1} = \phi_t \cup \{u'\}$. Instead of directly selecting the vertex with greatest probability, RL-QVO makes the selection according to the probabilities of vertices in the action space

to allow more exploration. Please note that the first vertex can also be selected according to the degree.

Reward Design. The reward is key factor in reinforcement learning, in this paper, the reward includes the reduced number of enumeration, validate reward for probability scores and entropy of the probabilities. One immediate reward is the reduced enumeration number $\Delta\#_{enum} = \#_{enum}(\phi) - \#_{enum}(\phi_{base})$, where ϕ is the learned order of RL-based agent and ϕ_{base} is the baseline order produced by existing subgraph matching algorithms. In our model, we use the state-of-the-art method Hybrid as our baseline algorithm. Specifically, according to [14], the ordering method of RI [16] has the best performance and is used in Hybrid. Therefore, we choose the order produced by RI as our baseline order, *i.e.,* $\phi_{base} = \phi_{RI}$. Considering the varying orders of magnitude of $\Delta\#_{enum}$ with different query graphs, the enumeration reward r_{enum} is defined as $r_{enum} = f_{enum}(\Delta\#_{enum})$, where $f_{enum}(\cdot)$ is a function such as logarithm which reduces the gaps (differences) between enumeration rewards under different query graphs to stabilize the computation. Intuitively, the model is more likely to gain greater enumeration reward r_{enum} on the complex queries which inherently require more rounds of enumeration procedure. Actually, the average enumeration time is usually dominated by the time costs of these complex queries. Therefore, the policy network pays greater importance to the complex queries with the hope to reduce more enumeration numbers in total. Since the enumeration reward $r_{enum,t}$ at step t cannot be determined until the final order ϕ is obtained, all rewards $r_{enum,t}$ at steps t share the same value as $r_{enum} = f_{enum}(\Delta\#_{enum})$. Meanwhile, this shared reward value enables the policy network to consider the long-term reward at every step, thus reduces the probability to fall into the local optimum.

We also design the step-wise validate rewards $r_{val,t}$. A small positive reward is assigned if the policy network produces a valid probability distribution, i.e., the vertex with largest probability is in the action space $u' \in N(\phi_t)$. Otherwise, a negative punishment is assigned which is greater than the positive reward in absolute value. Please note that even if the policy network produces invalid probabilities, our model still guarantees to generate a *connected order* ϕ by masking out the vertices that are not in the action space before making selection.

Furthermore, an entropy reward is considered at every step to encourage the model to output action probability distribution with *high* entropy [59]. Hence, the model is more likely to take actions unpredictably, which avoids the network converging too quickly on a policy that is locally optimal. This entropy reward $r_{h,t}$ is defined as $r_{h,t} = H(P_{\pi_\theta}(\phi_t, N(\phi_t))$, where $H(\cdot)$ is the entropy function, π_θ is a policy network with parameters θ, and $P_{\pi_\theta}(\phi_t, N(\phi_t))$ is the output probability at step t with given order ϕ_t and action space $N(\phi_t)$. Therefore, overall step-wise reward is formulated in the following Equation 1:

$$R_t = r_{enum} + \beta_{val} \cdot r_{val,t} + \beta_h \cdot r_{h,t}, \tag{1}$$

in which β_{val} and β_h denote the reward coefficients for validate reward and entropy reward respectively to tune their impact on training process.

In query ordering for subgraph matching task, the starting nodes in the order are usually more important than the trailing nodes. Therefore, when calculating the overall rewards for the policy network, we assign a decay factor to the step-wise rewards and formulate the overall rewards as follows:

$$R_{q,\theta} = \sum_{t=1}^{|V(q)|} \gamma^t R_t, \tag{2}$$

where $\gamma \in (0, 1)$ is a decay factor which enables the model to consider the importance of nodes in the learned order during training procedure.

D. RL-QVO Policy Network Architecture

In this section, we introduce the architecture of policy network in our RL-based method RL-QVO, which generates the query vertex order for subgraph matching.

Framework. Framework of RL-QVO is illustrated in Figure 2. RL-QVO first computes the vector representations $\boldsymbol{x}_u$ for query vertices $u \in V(q)$ by exploiting graph neural networks. The representation matrix of query vertices $\boldsymbol{X}_q$ is served as the input of a multi-layer perceptron (MLP) to obtain the probability distribution on action space for vertex selection.

Action Space. As introduced before, the action space $AS(t)$ is defined by the neighbor vertices set $N(\phi_t) = \{N(u) | \forall u \in \phi_t, N(u) \notin \phi_t\}$ of the ordered vertices at current step t to ensure the connectivity of generated orders. This constraint applies for most ordering methods in backtracking based subgraph matching algorithms. In the case that there is only one vertex in the action space, *i.e.,* $|AS(t)| = 1$, RL-QVO directly selects the only candidate as the next node without performing computation.

Policy Network. The policy network is a neural network which learns the rule for solving the target problem. With given state, a policy network returns a probability distribution over the action space. Then, according to the produced probability distribution, RL-QVO progressively selects the vertex to add into the matching order ϕ.

One naive solution is to use the simple multi-layer perceptron (MLP) as the policy network. However, MLP cannot capture the structural information with in the query graphs and usually has limited performance in complex tasks.

In our model, the policy network includes two main parts: the graph neural network that aggregates and extracts the graph information; and the multi-layer perceptron which finally produces the probability distribution.

In order to obtain the vector representations for query vertices, we utilize graph neural network, a well-studied technique which achieves the state-of-the-art performance in graph representation learning. We find in our experiment (see Section IV-D) that the performance of our model is not bound to the selection of GNN. Specifically, the policy network π is parameterized as graph convolutional network (GCN) [21] to embed the query vertices.

The high level idea of graph convolutional neural network is to perform a message passing along edges in the graph for total of $\mathcal{L}$ layers. Therefore, RL-QVO could not only obtain the heuristic information from the initial feature matrices, but also make prediction based on auxiliary structural and high-order neighboring information which are overlooked in

conventional subgraph matching algorithms. The aggregation of graph convolutional neural network can be formulated as follows:

$$\boldsymbol{H}^{(l+1)} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{3}$$

where $\boldsymbol{W}^{(l)}$ and $\boldsymbol{H}^{(l)}$ are the weight and the input feature matrices of lth layer respectively, $\boldsymbol{H}^{(0)}$ is the initial feature representation introduced in Section III-C. $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with self loops, $\tilde{\boldsymbol{D}}$ is a diagonal matrix where $\tilde{\boldsymbol{D}}_{ii} = \sum_i \tilde{\boldsymbol{A}}_{ij}$, and σ is an activation function such as ReLU. RL-QVO is compatible to any graph neural network. Because GCN could capture the structural, label and neighboring information with simple network structure and efficient computation process, we choose graph convolutional network as the graph representation learning module of RL-QVO. $\mathcal{L}$ layers of GCNs output the representations $\boldsymbol{H}_q^{\mathcal{L}}$ for all query vertices. RL-QVO then applies the multi-layer perceptron (MLP) on obtained representation $\boldsymbol{H}_q^{\mathcal{L}}$ to select the next node. Specifically, we use two linear neural layers with mask and normalization operations:

$$\mathbb{P}_{u'}^{(t)} = \pi(\cdot|S^{(t)}) = Softmax(mask_{u' \in AS(t)}(\boldsymbol{W}_2 \cdot \sigma(\boldsymbol{W}_1 \boldsymbol{h}_{u'}^{(t)}))), \tag{4}$$

Where $\boldsymbol{h}_{u'}^{(t)}$ is the vector embedding of query vertex u'. The output dimension of the MLP is 1, therefore, we get a real number score as the selection probability for each node. We utilize the mask operation to filter out the probability scores of vertices that are out of the action space, and then apply the softmax function on the candidate query vertices: $Softmax(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}_i}}{\sum_j (e^{\boldsymbol{z}_j})}$ to produce the normalized probabilities. These normalized probabilities for candidate vertices are also used to compute the entropy rewards mentioned in Section III-C. Please note that in Equation 4, we omit the bias for simplicity.

E. Policy Training

In the training phase, given N_T training query graphs $q = \{q_i | i = 1, ..., N_T\}$ and a data graph G, our goal is to maximize the expected rewards of policy network π_θ for the training graphs. As introduced in the previous subsection, the policy network is consist of layers of GCN and MLP. The reward of RL-QVO's policy network π_θ with parameters θ at time step t is the summation of rewards for all query graphs in the training batch as follows:

$$r_t(\theta) = \sum_{i=1}^{N_T} R_i(\phi_i; \theta_i), \tag{5}$$

where R_i is the reward for query graph q_i defined in Eq. 2. We utilize the proximal policy optimization (PPO) [60] to train the policy network. PPO is a policy gradient method which utilizes a sampling policy network that enables the model to update with sampled data in multiple epochs. In our case, the policy network $\pi_{\theta'}$ from the previous epoch (has not been updated) is used as the sampling policy network. The loss function is formulated as

$$J_r^{(t)}(\theta) = \sum_{(a_t, s_t)} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)} r_t(\theta), clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) r_t(\theta)), \tag{6}$$

$$J(\theta) = \sum_{t=1}^{|V(q)|} J_r^{(t)}(\theta), \tag{7}$$

where θ' is the parameters of policy network in previous epoch, ϵ is a factor to clip $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}$ which is the probability ratio of action a_t with state s_t computed by current and sampling policy networks. At every training step, the parameters from the previous epoch θ' remain fixed, and the parameters θ of the policy network are updated via backpropagation according to loss function described in Eq. (6), (7). The model replaces the parameters of sampling policy network with that of current policy network after each training epoch.

With the matching order obtained by the policy network, the algorithm executes the enumeration procedure to find all subgraph matches. We adopt the commonly used enumeration procedure summarized in Algorithm 2, which is also utilized in the state-of-the-art baseline method Hybrid.

F. Incremental Training

In experiment, we found that RL-QVO might have poor performance on small query sets, because the query sets with more vertices since these graphs usually require more enumeration number and potentially have greater values for objective function, which dominate the time cost of training procedure. Besides, more training time is required if we train RL-QVO on all query sets. Therefore, we incorporate the incremental training procedure which first trains RL-QVO on one query set for a large number of epochs and then trains RL-QVO on other query set by minimizing objective function in Eq. (7). In our experiment, we found a smaller value of training epochs (e.g., 10) is enough to prevent RL-QVO from catastrophic forgetting, improve the performance of RL-QVO on new query set and save the overall training time. Please refer to Section IV-F for more details.

G. Complexity Analysis

RL-QVO is built on the neural network framework that enjoys excellent time efficiency in terms of query process. With a given query graph q and data graph G, RL-QVO is only required to perform computation of graph neural networks and MLP on the query graph to produce the probability for node selection at every time step. In order to obtain the matching order for graph with $|V(q)|$ vertices, such computation should be executed for $|V(q)|$ times. Since the time complexity of GNN is $O(|E(q)|)$ [23] and that for MLP is $O(d^2)$ where d is the dimension of representations. As a result, the overall time complexity of RL-QVO for query vertex ordering is $O(|V(q)| \times (|E(q)| + d^2))$. Because the size of query graph q is relatively small, the time complexity of RL-QVO is negligible compared to the time cost of recursive enumeration. In terms of the space complexity, RL-QVO requires fixed space for the parameters, which is determined by the input and output vector dimensions of a network. Specifically, the space complexity is $O(\mathcal{L} \times d^2)$, where $\mathcal{L}$ is the number of neural layers and d is the dimension of representations. Therefore, our proposed RL-QVO remains fixed space requirement with growing sizes of query and data graphs.

H. Discussion

In the training phase of this work, we adopt the Proximal Policy Optimization (PPO) to train our policy network. Though PPO outperforms other reinforcement learning training methods, such as actor-critic and Q-learning in this work, it also requires the matching on the training instances, which results in extra training time. The training time is significantly reduced by incremental training and reducing number of enumerated matches in the training phase. This overhead could be entirely avoided in other reinforcement learning frameworks. This could be an opportunity for the future work.

IV. Experiment

This section shows the results of empirical studies.

A. Experiment Setup

Compared Methods. In order to show the efficiency and effectiveness of our proposed RL-QVO, we conduct the experiments on the following compared methods including Hybrid whose techniques are recommended in [14].

- **QuickSI (QSI)** [15] is a directly enumeration method which does not generate candidate set, but filters out the unpromising vertices during enumeration. QSI uses the *infrequent-edge first ordering method*.
- **RI** [16] is a state space representation based model which generate the query vertex order only based on the structure of query graph q.
- **VF2++** [17] is also a state space representation based model which generates the query vertex order according to the node label frequency.
- **VEQ** [20] is a recent published method with state-of-the-art performance on both subgraph query and subgraph matching tasks.
- **Hybrid** [14] has a combination of candidate filtering, vertex ordering and enumeration methods of GQL, RI and QSI [61] respectively. These techniques are proven to have the best performance according to an extensive empirical study [14].
- **RL-QVO** is our proposed backtracking based subgraph matching algorithm. Particularly, the proposed RL-based method is utilized to generate the matching order. Regarding the candidate set generation and enumeration procedure, we use the same implementations of Hybrid.

We obtained the code of VEQ [*] from the authors of [20]. All other baseline methods are implemented by the authors of [14] in C++ [*]. The machine learning part of RL-QVO is implemented using Pytorch which could be computed on GPUs, while the candidate generation and enumeration methods are adapted from the code of [14] in C++.

Experiment Environment. We conducted the experiments on the servers running RHEL 7.7 system, which have Intel Xeon Gold 6238R 2.2GHz 28cores CPU and NVIDIA Quadro RTX 5000 GPU with 88GB RAM (Six Channel).

Data Graph. We conducted the experiments on six real-life datasets. The number of vertices varies from $3,112$ to $1,134,890$.

[*]https://github.com/SNUCSE-CTA/VEQ

[*]https://github.com/RapidsAtHKUST/SubgraphMatching

TABLE II: Datasets Properties

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	d
Citeseer	3,327	4,732	6	1.4
Yeast	3,112	12,519	71	8.0
DBLP	317,080	1,049,866	15	6.6
Youtube	1,134,890	2,987,624	25	5.3
Wordnet	76,853	120,399	5	3.1
EU2005	862,664	16,138,468	40	37.4

TABLE III: Query Sets

Dataset	Query Set	Default
Citeseer, Yeast, DBLP, Youtube, EU2005	Q_4, Q_8, Q_{16}, Q_{32}	Q_{32}
Wordnet	Q_4, Q_8, Q_{16}	Q_{16}

We obtain the datasets used by previous works [11], [14]. Six real-life graphs can be classified into five different categories: Citeseer is a citation network, Yeast is a biology network, DBLP and Youtube are social networks, Wordnet is a lexical network and EU2005 is a web network. The detailed properties of the datasets are summarized in Table II.

Query Graph. As described in [14], the query graphs are generated for each data graph by randomly extracting **connected** subgraphs from G. We generate the query set for Citeseer following the setting introduced in [14]. The number of query vertices $|V(q)|$ varies from 4 to 32 for Citeseer, Yeast DBLP, Youtube and EU2005; $|V(q)|$ varies from 4 to 16 for Wordnet. There are 400 query graphs in Q_8 and Q_{16}, and 200 query graphs in Q_4 and Q_{32}. 50% of the query graphs are used for training, and the remaining query graphs are used for evaluation. Table III lists the details of query sets for each dataset, where Q_i denotes the query set in which the query graphs have i vertices. Due to the space limit, by default, we report the time cost results on query graph set with 16 vertices, *i.e.,* Q_{16} for Wordnet, and query set with 32 vertices, *i.e.,* Q_{32} for other data graphs as representatives.

Experiment settings. In this experiment, RL-QVO has two layers of graph convolutional networks to obtain the representations for the query nodes. After GCN layers, there is a two-layer linear neural network that produces the selection probability for each node. The learning rate is set to 10^{-3}, the output dimension of GCN is set to 64, number of training epochs is 100 by default and 10 for incremental training. All the scaling factors α are set to 1. We also apply a dropout ratio at 0.2 during training. Due to the enormous time cost to find all subgraph matches, existing works [14], [20] measure the matching time to find the first 10^5 matches. In this work, unless otherwise stated, we follow the settings of the existing works and terminate the enumeration procedure when 10^5 matches are found during training and evaluation phases for fair comparison and reducing overall time cost. We set a time limit 500 seconds for subgraph matching, if the query process exceeds the time limit during training, we skip this query graph for training to save experimental time. During evaluation, if a compared algorithm cannot finish the query within the time limit, we denote the query graph as a unsolved query and assign the time cost as 500 secs for this query. If a query graph remains unsolved by all compared methods, we would exclude this query graph in computing the average query processing time and enumeration time. We also count the numbers of unsolved queries to compare the capabilities in solving the worse case queries for all compared methods. The results are shown in Section IV-B.

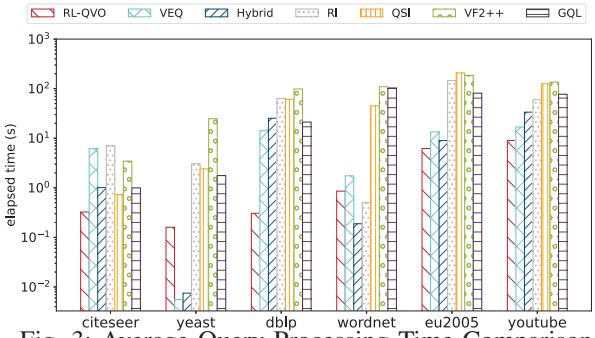


Fig. 3: Average Query Processing Time Comparison

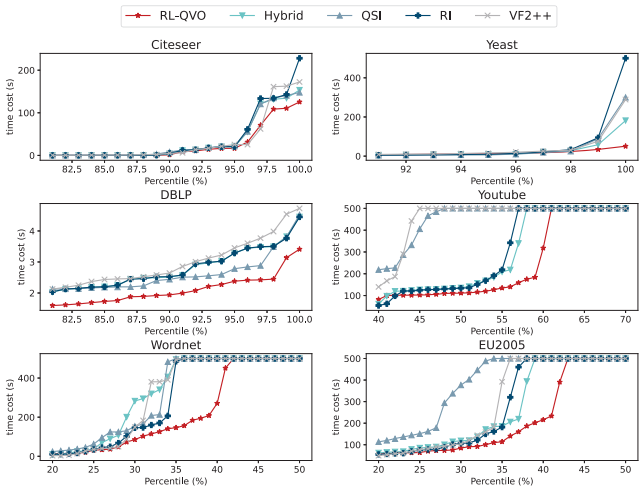


Fig. 4: Query Processing Time Percentile Comparison

Evaluation Metrics. In this experiment, we evaluate the time efficiency of compared methods. We use the average query processing time (i.e., the query response time) and enumeration time (i.e., the time cost of enumeration procedure) for the comparison. Recall that we say a query is unsolved if it cannot be finished within 500 seconds. We also report the training time and the memory consumption of the our model.

B. Query Processing Time Comparison

In the first set of experiments, we evaluate the query processing time for all compared subgraph matching algorithms with default query graph sets, where 6 real-life graphs are deployed. Note that the query processing time t includes the filtering methods time cost t_{filter}, the ordering time t_{order} and enumeration time t_{enum}, i.e., $t = t_{filter} + t_{order} + t_{enum}$.

Average Query Processing Time. We first report the average query processing time for the given query graphs on default query sets, and the results are illustrated in Figure 3. It is shown that RL-QVO generally outperforms the competitors because of the high quality matching order obtained by our advanced model. In particular, RL-QVO outperforms VEQ by up to two orders of magnitude on Citeseer and DBLP. RL-QVO is also more than two orders of magnitude faster than Hybrid on DBLP. In the experiments of Section IV-C, we further justify that this is a significant achievement where the optimal matching order is considered.

Cumulative Query Processing Time Distribution. To better understand the query processing time distribution, Fig. 4 details the query processing time comparison by showing a cumulative distribution of the query processing time for all compared methods. Please note that in order to demonstrate the differences much clearer, the query processing time report in this experiment is the time cost to find *all* subgraph matches in the corresponding data graphs, which is much greater than the time cost to find first 10^5 matches. The gaps of time cost between RL-QVO and other compared methods grow with percentile, which shows the efficiency of RL-QVO in handling the hard queries. This is a big advantage of RL-QVO compared to other competitors because the response time of queries at high percentile (*i.e.,* hard queries) is critical for the throughout of the system in many industry applications.

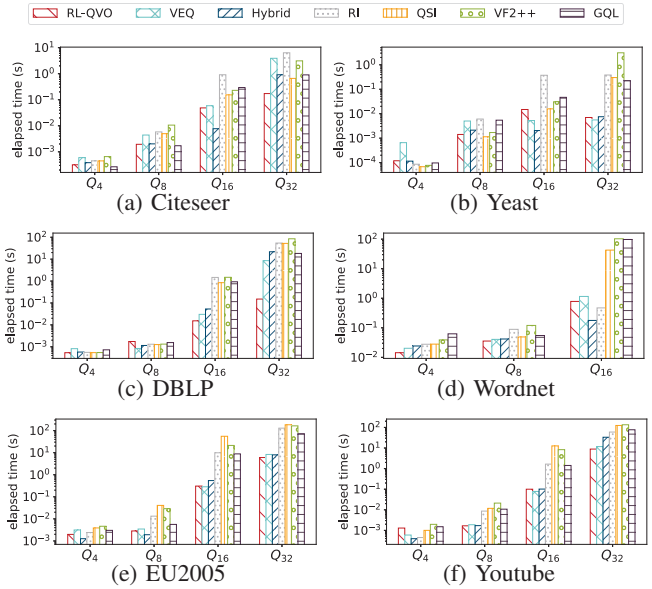


Fig. 5: Enumeration time cost comparison

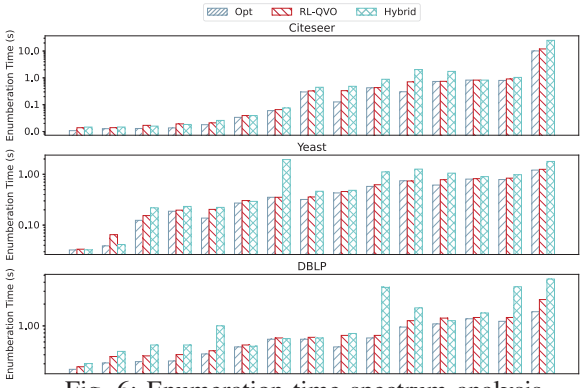


Fig. 6: Enumeration time spectrum analysis with optimal orders

Number of Unsolved Queries. Fig. 4 also demonstrates the number of unsolved query graphs with default query sets for Youtube, Wordnet and EU2005 data graphs. Note that we set the query time of each unsolved query to 500 seconds (preset time limit). It is shown that RL-QVO has much less number of unsolved queries compare to other algorithms.

C. Enumeration Time Comparison

The enumeration time is the determining factor of the overall time cost of subgraph matching process, and also directly reflects the qualities of matching orders generated

by different algorithms. We compare the enumeration time of VEQ, Hybrid, QSI, RI, VF2++ and GQL with RL-QVO to examine the qualities of the matching orders generated by these methods. Since all these methods utilize the same enumeration methods which are implemented in the same way, the enumeration time costs could directly reflect the qualities of the output matching orders.

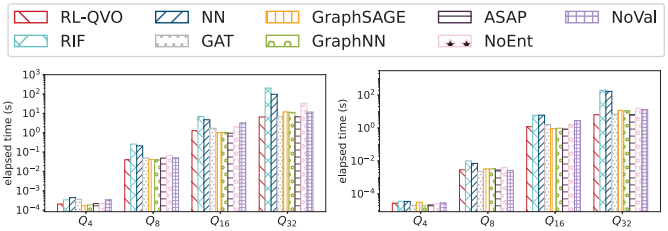
Average Enumeration Time. The average enumeration time with varying query sizes are shown in Fig. 5. It is reported that our proposed RL-QVO has the state-of-the-art performance on all datasets with all query sizes. Similar as query processing time, our proposed RL-QVO could improve the enumeration time up to 2 orders of magnitude compared to the *best* performed baseline method on Q_{32} of DBLP. The performance gaps between RL-QVO and other baseline methods become much more significant with the growth of query vertex numbers, which indicates that RL-QVO has formidable ability in handling ordering problem in large search space. It is also worth noting that RL-QVO achieves similar performance as baseline algorithms w.r.t the enumeration time on Yeast data graph, which is much better than the time cost for query processing. This result indicates that RL-QVO might need relatively more time to generate matching order, but the generated order has high quality even on small data graphs like Yeast.

Comparison with Optimal Matching Order. To better investigate the goodness and the potential improvement space of the matching orders obtained by the algorithms, we also consider the optimal matching orders in the experiments. As it is time prohibitive to find the optimal matching orders for large size queries and data graphs. We only consider the settings where there are no unsolved queries for RL-QVO and Hybrid. Particularly, we conduct the spectrum analysis on Citeseer, Yeast and DBLP datasets to find *all* subgraph matches with 15 randomly selected query graphs with 8 vertices (Q_8) for each dataset. To obtain the *optimal matching order*, we generate the orders of all permutations of the query vertices, and feed them into the subgraph matching algorithm with the same filtering and enumeration methods as RL-QVO and Hybrid. We pick the permutation that requires the minimum enumeration number as the *optimal matching order*.

We compare the enumeration time of RL-QVO and Hybrid to the optimal matching order based algorithm, denoted by **Opt**. Note that all three algorithms use the same filtering and enumeration implementations. The analysis results are illustrated in Fig. 6 where the bars indicate the enumeration time costs for the optimal order, RL-QVO's order and Hybrid's order. The results show that compared to the ordering methods of Hybrid, RL-QVO is more likely to generate *near optimal* orders for matching, thus have better overall search performance. Moreover, considering the gaps between *Opt* and Hybrid for these queries in Fig. 6, we boast that that RL-QVO makes a significant improvement in terms of enumeration time (*i.e.,* the quality of the matching order).

D. Ablation Study

We evaluate the effectiveness and efficiency of different graph neural networks in this ablation study. We change the agent components of RL-QVO to get the following variants:



(a) Query Processing Time (b) Enumeration Time

Fig. 7: Time comparison for ablation study on EU2005.

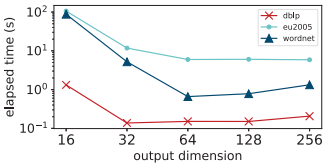


Fig. 8: Query processing time vs. output dimension

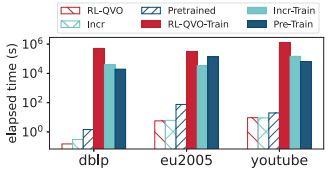


Fig. 9: Query processing time and training time comparison of incremental training.

RL-QVO-RIF is the variant that each vertex has random input feature rather than our carefully designed features.
RL-QVO-NN is the variant that replaces the GCN with naive multi-layer perceptron (MLP) as the policy network, which tests the effectiveness of the GCN in this ordering task.
RL-QVO-GAT is the variant that uses graph attention network [22] as the policy network.
RL-QVO-GraphSAGE is the variant that uses GraphSAGE [57] as the policy network.
RL-QVO-GraphNN uses the GNN operator from [62] as the policy network.
RL-QVO-ASAP exploits the GNN operator in ASAP [63] as the policy network.
RL-QVO-NoEnt is the variant without the entropy reward.
RL-QVO-NoVal is the variant without the validation reward.

In this experiment, we compare the query processing and enumeration time of RL-QVO and its variants on data graph EU2005 with query sets Q_4, Q_8, Q_{16} and Q_{32}. The results are illustrated in Fig. 7. Specifically, the significant performance difference between RL-QVO and RL-QVO-NN validates the effectiveness of graph neural network in our model. It can also be concluded from the results that different GNN models has limited differences w.r.t the enumeration time, i.e., the performance of RL-QVO is not bound to the selection of GNN model. The time cost gap between RL-QVO-RIF and RL-QVO proves the effectiveness of our carefully designed feature. Furthermore, the time difference between RL-QVO-NoEnt/NoVal and RL-QVO shows the necessity of the entropy and validate rewards, especially with the large-size query sets.

E. Query Processing Time vs. Output Dimension

In this subsection, we analyze how the output dimension of RL-QVO influence the query processing time. We vary the output dimension of RL-QVO in $\{16, 32, 64, 128, 256\}$ and conduct the experiment on DBLP, EU2005 and Wordnet with their default query sets. The results are summarized in Fig. 8. When the output dimension is 32, RL-QVO has limited performance because of the small parameter space. With the growth of output dimension, the query processing time reduces, and the salient point is around 64. When the

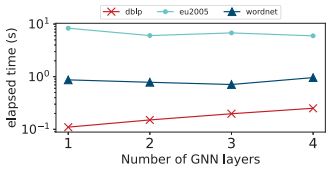


Fig. 10: Query processing time vs. number of GNN layers

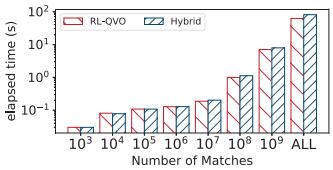


Fig. 11: Enumeration time vs. number of matches.

dimension continues growing, the query processing time will increase because the time cost for ordering t_{order} increases with the growth of output dimension.

F. Training Time and Order Inference Time

In this subsection, we evaluate the query processing time and training time of RL-QVO to show the efficiency brought by the incremental training.

As mentioned in Section III-F, RL-QVO could incorporate the incremental training to reduce the overall training time. We compare three training method in this experiment. (1) We train the model with the default query set for 100 epochs on corresponding data graphs. (2) For each data graph, we first train the model on query set with less query vertices like Q_8 (for Wordnet) or Q_{16} (for other data graphs) for 100 epochs. Then incremental training is applied on default query set for 10 epochs. (3) The pre-trained model is directly applied on default query set. The results are illustrated in Fig. 9. Model trained on default query set for 100 epochs (denoted as RL-QVO in Fig. 9) always has the least query processing time. Directly applying the pre-trained model (denoted as Pretrained) usually has significant performance gap with RL-QVO. Compared with above training methods, the incremental trained model (denoted as Incr) could save nearly two orders of magnitude of training time while only sacrificing negligible performance. We would like to emphasize that all the time cost results reported in previous experiments are produced by incremental training method.

Order inference time. As analyzed in Section III-G, the computational complexity of RL-QVO for query vertex ordering generation is $O(|V(q)| \times (|E(q)| + d^2))$. In our experiments, RL-QVO could produce the matching order within $100ms$.

G. Query Processing Time vs. Number of GNN layers

We analyse how the number of GNN layers influences the query processing time in this subsection. We vary the number of GNN layers in $\{1, 2, 3, 4\}$ and conduct the experiment on on DBLP, EU2005 and Wordnet with their default query sets. The results are illustrated in Fig. 10. When the data graph is relatively small, *e.g.,* DBLP, the query processing time increases with the number of GNN layers near-linearly, since the query processing time is mainly consist of the time to generate the matching order. Otherwise, on the lager data graphs, the model with only one GNN layer usually has the worst performance due to its limited power to leverage the structural information in the graphs. When the number of layers is more than one, there is no significant difference between the query processing time, except the time cost for model with 4 layers on Wordnet.

TABLE IV: Space Evaluation

Dataset	Graph Space	Model Space
Citeseer	112.4 kB	186.2 kB
Yeast	260.8 kB	186.2 kB
DBLP	30.4 MB	186.2 kB
Youtube	89.7 MB	186.2 kB
Wordnet	3.5 MB	186.2 kB
EU2005	437.6 MB	186.2 kB

H. Enumeration Time vs. Number of Matches

As mentioned in Section IV-A, we measure the matching time to find the first 10^5 matches in most of previous experiments. In this subsection, we report the average enumeration time of RL-QVO and Hybrid varying the number of matches enumerated on data graph Youtube with Q_{16}. The results are shown in Fig. 11, where *ALL* denotes the experiment setting that finds all the matches of a query graph in the data graph. Please note that the processing time of query that exceeds the time limit (500 secs) is set to 500 seconds, and the time costs of queries that cannot be processed by both methods within the time limit are not reported in this experiment. As illustrated in Fig. 11, when the number of matches is relatively small (from 10^3 to 10^6), there is no notable time difference between RL-QVO and Hybrid. With the growth of number of enumerated matches, RL-QVO has more significant advancement compared to Hybrid in terms of the enumeration time, which indicates the superior order generation ability of RL-QVO over Hybrid on large search space.

I. Space Evaluation

As discussed in Section III-G, the space complexity for RL-QVO is fixed with growing sizes of query and data graphs. Here, we report the exact space requirements for storing the parameters of the network and corresponding data graphs. The results are demonstrated in Table IV. Even for data graph like EU2005 which requires 437.6 MB to store, RL-QVO only needs 186.2 kB to save parameters while achieving outstanding performance in query vertex ordering. In terms of the memory consumption, RL-QVO might require more space for query graphs, however, the memory cost is still dominated by the parameter space of the model, which is relatively small.

V. Conclusion

Subgraph matching is a fundamental research topic in database and data mining communities, which is a NP-Complete problem. One important branch of the subgraph matching algorithms follows the backtracking search paradigm, and it is shown that the search performance is heavily affected by the quality of the matching order used in the search. We notice that existing ordering methods with heuristic strategies are lacking the capability to fully exploit the structural and label information to generate high-quality matching order (i.e., query vertex order). In this paper, we proposed RL-QVO, a reinforcement learning-based model for query vertex ordering. RL-QVO learns the policy to generate the matching order considering both graph structure information and the long-term benefits. Extensive experiments prove the efficiency of RL-QVO.

Acknowledgement. Y. Zhang is supported by ARC DP210101393, L. Qin is supported by ARC FT200100787 and DP210101347, W. Wang was supported by HKUST(GZ) Grants G0101000028 and GZU22EG04, W. Zhang is supported by FT210100303 and DP200101116

References

[1] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. VLDB Endow.*, vol. 11, no. 4, pp. 420–431, 2017.

[2] N. Przulj, D. G. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks," *Bioinform.*, vol. 22, no. 8, pp. 974–980, 2006.

[3] T. A. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock, "New specifications for exponential random graph models," *Sociological methodology*, vol. 36, no. 1, pp. 99–153, 2006.

[4] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 335–346.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] L. Lai, L. Qin, X. Lin, and L. Chang, "Scalable subgraph enumeration in mapreduce," *Proc. VLDB Endow.*, vol. 8, no. 10, pp. 974–985, 2015.

[7] L. Lai, L. Qin, X. Lin, Y. Zhang, and L. Chang, "Scalable distributed subgraph enumeration," *Proc. VLDB Endow.*, vol. 10, no. 3, pp. 217–228, 2016.

[8] L. Lai, Z. Qing, Z. Yang, X. Jin, Z. Lai, R. Wang, K. Hao, X. Lin, L. Qin, W. Zhang, Y. Zhang, Z. Qian, and J. Zhou, "Distributed subgraph matching on timely dataflow," *Proc. VLDB Endow.*, vol. 12, no. 10, pp. 1099–1112, 2019.

[9] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proc. VLDB Endow.*, vol. 5, no. 9, pp. 788–799, 2012.

[10] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 804–818, 2017.

[11] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, F. Özcan, G. Koutrika, and S. Madden, Eds. ACM, 2016, pp. 1199–1214.

[12] M. Han, H. Kim, G. Gu, K. Park, and W. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds. ACM, 2019, pp. 1429–1446.

[13] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 337–348.

[14] S. Sun and Q. Luo, "In-memory subgraph matching: An in-depth study," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1083–1098.

[15] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 364–375, 2008.

[16] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC bioinformatics*, vol. 14, no. 7, pp. 1–13, 2013.

[17] A. Jüttner and P. Madarasi, "Vf2++—an improved subgraph isomorphism algorithm," *Discrete Applied Mathematics*, vol. 242, pp. 69–81, 2018.

[18] H. He and A. K. Singh, "Graphs-at-a-time: query language and access methods for graph databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed. ACM, 2008, pp. 405–418.

[19] B. Bhattarai, H. Liu, and H. H. Huang, "CECI: compact embedding cluster index for scalable subgraph matching," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds. ACM, 2019, pp. 1447–1462.

[20] H. Kim, Y. Choi, K. Park, X. Lin, S.-H. Hong, and W.-S. Han, "Versatile equivalences: Speeding up subgraph query processing and subgraph matching," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 925–937.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[22] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[24] J. Lee, W. Han, R. Kasperovics, and J. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 133–144, 2012.

[25] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 459–468.

[26] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, 2004.

[27] V. Carletti, P. Foggia, and M. Vento, "VF2 plus: An improved version of VF2 for biological graphs," in *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, ser. Lecture Notes in Computer Science, C. Liu, B. Luo, W. G. Kropatsch, and J. Cheng, Eds., vol. 9069. Springer, 2015, pp. 168–177.

[28] Z. Yang, L. Lai, X. Lin, K. Hao, and W. Zhang, "Huge: An efficient and scalable subgraph enumeration system," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2049–2062.

[29] H. Q. Ngo, "Worst-case optimal join algorithms: Techniques, results, and open problems," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, J. V. den Bussche and M. Arenas, Eds. ACM, 2018, pp. 111–124.

[30] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn, "Design and implementation of the logicblox system," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds. ACM, 2015, pp. 1371–1382.

[31] S. Zampelli, Y. Deville, and C. Solnon, "Solving subgraph isomorphism problems with constraint programming," *Constraints*, vol. 15, no. 3, pp. 327–353, 2010.

[32] C. McCreesh, P. Prosser, and J. Trimble, "The glasgow subgraph solver: using constraint programming to tackle hard subgraph isomorphism problem variants," in *International Conference on Graph Transformation*. Springer, 2020, pp. 316–324.

[33] C. McCreesh, P. Prosser, C. Solnon, and J. Trimble, "When subgraph isomorphism is really hard, and why this matters for graph databases," *Journal of Artificial Intelligence Research*, vol. 61, pp. 723–759, 2018.

[34] S. Zhang, S. Li, and J. Yang, "Gaddi: distance index based subgraph matching in biological networks," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, 2009, pp. 192–203.

[35] P. Zhao and J. Han, "On graph query optimization in large networks," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 340–351, 2010.

[36] C. R. Rivero and H. M. Jamil, "Efficient and scalable labeled subgraph matching using sgmatch," *Knowledge and Information Systems*, vol. 51, no. 1, pp. 61–87, 2017.

[37] F. Katsarou, N. Ntarmos, and P. Triantafillou, "Performance and scalability of indexed subgraph query processing methods," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1566–1577, 2015.

[38] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," *Proceedings of the VLDB Endowment*, vol. 6, no. 2, pp. 133–144, 2012.

[39] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec *et al.*, "Neural subgraph matching," *arXiv preprint arXiv:2007.03092*, 2020.

[40] X. Liu, H. Pan, M. He, Y. Song, X. Jiang, and L. Shang, "Neural subgraph isomorphism counting," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1959–1969.

[41] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" *arXiv preprint arXiv:2002.04025*, 2020.

[42] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6351–6361.

[43] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving sat solver heuristics with graph networks and reinforcement learning," *arXiv preprint arXiv:1909.11830*, 2019.

[44] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 537–546.

[45] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 15 554–15 566.

[46] C. Ren, L. An, Z. Gu, Y. Wang, and Y. Gao, "Rebalancing the car-sharing system with reinforcement learning," *World Wide Web*, vol. 23, no. 4, pp. 2491–2511, 2020.

[47] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.

[48] S. Hu, Z. Xiong, M. Qu, X. Yuan, M. Côté, Z. Liu, and J. Tang, "Graph policy network for transferable active learning on graphs," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[49] J. You, B. Liu, Z. Ying, V. S. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 6412–6422.

[50] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds. ACM, 2019, pp. 750–760.

[51] H. Wang, D. Lian, Y. Zhang, L. Qin, and X. Lin, "Gognn: Graph of graphs neural network for predicting structured entity interactions," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1317–1323.

[52] Y. Hao, X. Cao, Y. Fang, X. Xie, and S. Wang, "Inductive link prediction for nodes having only attribute information," *arXiv preprint arXiv:2007.08053*, 2020.

[53] Y. Hao, X. Cao, Y. Sheng, Y. Fang, and W. Wang, "Ks-gnn: Keywords search over incomplete graphs via graphs neural network," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[54] H. Wang, D. Lian, Y. Zhang, L. Qin, X. He, Y. Lin, and X. Lin, "Binarized graph neural network," *World Wide Web*, vol. 24, no. 3, pp. 825–848, 2021.

[55] H. Wang, D. Lian, W. Liu, D. Wen, C. Chen, and X. Wang, "Powerful graph of graphs neural network for structured entity analysis," *World Wide Web*, pp. 1–21, 2021.

[56] C. Huang, Y. Fang, X. Lin, X. Cao, and W. Zhang, "Able: Meta-path prediction in heterogeneous information networks," *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 4, jan 2022.

[57] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[58] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[59] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[60] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[61] T. L. Veldhuizen, "Leapfrog triejoin: A simple, worst-case optimal join algorithm," *arXiv preprint arXiv:1210.0481*, 2012.

[62] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[63] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5470–5477.