

Approximate Subgraph Matching with Neural Graph Representations and Reinforcement Learning

Kaiyang Li, Shihao Ji, *Senior Member, IEEE*, Zhipeng Cai, *Fellow, IEEE*, and Wei Li

Abstract—Approximate subgraph matching (ASM) is a task that determines the *approximate* presence of a given query graph in a large target graph. Being an NP-hard problem, ASM is critical in graph analysis with a myriad of applications ranging from database systems and network science to biochemistry and privacy. Existing techniques often employ heuristic search strategies, which cannot fully utilize the graph information, leading to sub-optimal solutions. This paper proposes a Reinforcement Learning based Approximate Subgraph Matching (RL-ASM) algorithm that exploits graph transformers to effectively extract graph representations and RL-based policies for ASM. Our model is built upon the branch-and-bound algorithm that selects one pair of nodes from the two input graphs at a time for potential matches. Instead of using heuristics, we exploit a Graph Transformer architecture to extract feature representations that encode the full graph information. To enhance the training of the RL policy, we use supervised signals to guide our agent in an imitation learning stage. Subsequently, the policy is fine-tuned with the Proximal Policy Optimization (PPO) that optimizes the accumulative long-term rewards over episodes. Extensive experiments on both synthetic and real-world datasets demonstrate that our RL-ASM outperforms existing methods in terms of effectiveness and efficiency. Our source code is available at <https://github.com/KaiyangLi1992/RL-ASM>.

Index Terms—Approximate subgraph matching, Reinforcement learning, Graph Transformer

I. INTRODUCTION

Graph analysis, a sub-field of data mining, has gained popularity in recent years as graph structured data becomes increasingly ubiquitous in a broad range of domains [1], [2]. One of the fundamental problems of graph analysis is subgraph matching, which identifies the presence of a query graph in a large target graph. Subgraph matching has a wide variety of applications, including database retrieval [3], knowledge graph mining [4], biomedical analysis [5], social group finding [6], and privacy protection [7].

In most of the subgraph matching studies, the basic assumption is that graphs are noise-free and accurate. In order to identify an occurrence of a query graph, all nodes and edges of the query graph must occur in the target graph. In other words, the matching has to be exact. However, noise commonly exists in many real-world subgraph matching scenarios. Therefore, the Approximate Subgraph Matching (ASM) has emerged as a more practical task in graph analysis. For example, noise is prevalent in protein-protein interaction (PPI) networks [8] due

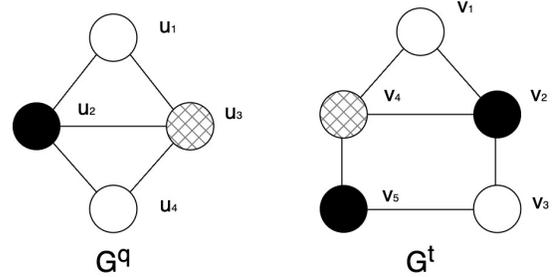


Fig. 1: An example of approximate subgraph matching, where $\{(u_1, v_1), (u_2, v_2), (u_3, v_4), (u_4, v_5)\}$ achieves the best approximate matching from query graph G^q to target graph G^t with the smallest graph edit distance of 1.

to errors in data collection and varying experimental thresholds. By discovering and analyzing approximate matches, biologists can still validate their hypotheses based on noisy protein interaction network data. Another example is social network de-anonymization [9], which involves identifying anonymous individuals within a social network by correlating nodes from an auxiliary network with those from a target network, where the auxiliary network is typically a noisy subgraph. Effective matching between these two graphs enables adversaries to launch de-anonymization attacks. Therefore, research into ASM is important to evaluate data vulnerability and develop privacy-preserving solutions.

Formally, given a query graph G^q , ASM aims to find a subgraph in target graph G^t that has the smallest graph edit distance (GED) to G^q . An example is illustrated in Fig. 1, where the mapping $\{(u_1, v_1), (u_2, v_2), (u_3, v_4), (u_4, v_5)\}$ achieves the best approximate match from G^q to G^t with the smallest GED of 1. The existing ASM algorithms can be mainly categorized into two classes: (1) the methods that convert ASM to exact subgraph matching [10], [11], and (2) the methods that employ a branch-and-bound algorithm for combinatorial optimization [12]. The first category of methods specifies a threshold k and reduces the ASM problem to identifying all exact matches between any prototype graphs and target graph, where the prototype graphs are all graphs that can be derived from the query graph with the GED smaller than k . Therefore, the exact matches of the prototype graphs are the approximate matches of the query graph. However, these methods only consider prototypes by adding edges to query graphs, excluding those that are derived by altering nodes' labels. This limitation renders these methods unsuitable for scenarios where nodes' labels are noisy. One of the reasons why these methods exclude prototype graphs generated by altering nodes' labels is that doing so would significantly increase the number of prototype graphs. For example, consider a graph G^q , which is a cycle

Kaiyang Li and Shihao Ji are with the University of Connecticut, 352 Mansfield Road, Storrs, CT 06269, USA (e-mail: kaiyang.li@uconn.edu; shihao.ji@uconn.edu).

Zhipeng Cai and Wei Li are with Georgia State University, 33 Gilmer Street SE, Atlanta, GA 30303, USA (e-mail: zcai@gsu.edu; wli28@gsu.edu).

graph C_8 , where each node has 16 possible label categories. With a GED threshold $k = 3$, considering only edge-induced distances, it results in 1,351 prototype graphs. However, if both edges and node labels are considered in GED, this number escalates to 347,971.

On the other hand, the branch-and-bound based ASM methods match one pair of nodes from query graph and target graph one at a time and extend intermediate results iteratively. For example, Tu et al. [12] calculate the GED lower bounds of the best possible solutions within each branch of the search tree, then compare these lower bounds with the GED of the current best match, and prune the branches that are not possible to yield an optimal solution, i.e., the branches with lower bounds greater than the GED of the current best match. However, these methods select mapping node pairs with a greedy strategy based on heuristics, i.e., selecting the node pair that leads to the branch with the minimized GED lower bound at each step by utilizing nodes' local structural and label information. Therefore, let alone its greedy nature, the method lacks the ability of fully utilizing the graph information in the two input graphs, leading to sub-optimal solutions.

To address these limitations of the existing approaches, we introduce a Reinforcement Learning based Approximate Subgraph Matching (RL-ASM) method. RL-ASM employs a Graph Transformer [13] to extract feature representations from graphs, whose performances in graph representation learning have been extensively validated both theoretically and empirically, allowing the use of full graph information for ASM. To mitigate the issues induced by greedy algorithms, we train our neural network based agent with RL algorithms to optimize an accumulative long-term reward over episodes. As a result, our RL-ASM achieves the solutions that are closer to the optimal ones, outperforming the existing search algorithms by a significant margin. The contributions of our work are summarized as follows:

- To the best of our knowledge, RL-ASM is the first work that leverages reinforcement learning for high-quality approximate subgraph matching.
- Our RL-based Graph Transformer model can fully utilize the graph information and select the node pairs by optimizing a long-term reward instead of being greedy.
- Extensive experiments conducted on synthetic and real-world graph datasets demonstrate that RL-ASM outperforms existing methods in terms of effectiveness and efficiency.

II. RELATED WORK

Traditional ASM Methods. Being an NP-hard problem [14], ASM has been tackled in various approaches. Tong et al. [15] investigate ASM in social networks and propose a method that samples the matched subgraph via random walk. Tian et al. [5] and Yuan et al. [16] break the query graph into small fragments and assemble the matches of these fragments to generate the entire match of the query graph. Other works [17]–[19] utilize the chi-square statistics to measure the node similarity based on the label distribution of the node's neighbors and apply the similarity measure to search for ASM. Sussman et al. [20] propose to calculate

the permutation matrix on adjacency matrix and match query graph to target graph via the permutation matrix. Recently, Reza et al. [10] and Zhang et al. [11] identify all prototype graphs whose distances from query graph are below a specified threshold, then reduce the ASM problem to the exact subgraph matching problem by searching for exact matches between the prototype graphs and target graphs. Tu et al. [12] approach the ASM problem by formulating it as a tree search problem, employing lower bounds and cutoffs to reduce the search space. Given a sufficient running time, this method can find an optimal solution, i.e., the subgraph of target graph that is the most similar to query graph.

Unfortunately, most of the existing methods utilize hand-crafted features to search for the mapped node pairs, and cannot fully exploit the graph information for matching. What's more, they all exploit greedy algorithms that overlook the potential better matches in future steps, leading to sub-optimal solutions.

Reinforcement Learning for NP-hard Graph Problems.

There are many prior works focusing on RL algorithms to solve NP-hard problems on graphs, including Minimum Vertex Cover [21], Network Dismantling [22], and Maximum Common Subgraph detection [23], [24]. As of exact subgraph matching, Wang et al. [25] utilize RL to determine the node mapping order on query graphs, while Bai et al. [26] utilize RL to determine the node mapping order on target graphs. Compared with exact subgraph matching, ASM represents an even more challenging task since ASM allows discrepancies between query graph G^q and the matched subgraph of G^t . Therefore, hard constraints that typically can be used to reduce the search space (e.g., requiring mapped nodes to have the same label) are not applicable. To address this, our RL-ASM employs a branch-and-bound algorithm to reduce the search space and leverages a Graph Transformer to extract graph features and select actions from a large action space.

III. PRELIMINARY

A. Problem Definition

Let $G = (V, E)$ denote a graph with a set of nodes V connected by a set of edges E . The approximate subgraph matching problem can be defined as follows.

Definition 1: Approximate Subgraph Matching (ASM): Given a query graph $G^q = (V^q, E^q)$ and a target graph $G^t = (V^t, E^t)$, ASM aims to identify a one-to-one mapping $M : V^q \rightarrow V^t$ such that the graph edit distance $C(M; G^q, G^t)$ is minimized, where $C(M; G^q, G^t)$ measures the discrepancies between query graph G^q and the subgraph of G^t induced by $M(V^q)$.

Following [12], [27], we adopt the graph edit distance $C(M; G^q, G^t)$ over nodes and edges as

$$C(M; G^q, G^t) = \sum_{u \in V^q} D_V(u, M(u)) + \sum_{e \in E^q} D_E(e, M(e)), \quad (1)$$

where the node distance is

$$D_V(u, M(u)) = \begin{cases} 0 & \mathcal{A}(u) = \mathcal{A}(M(u)) \\ 1 & \mathcal{A}(u) \neq \mathcal{A}(M(u)) \end{cases}, \quad (2)$$

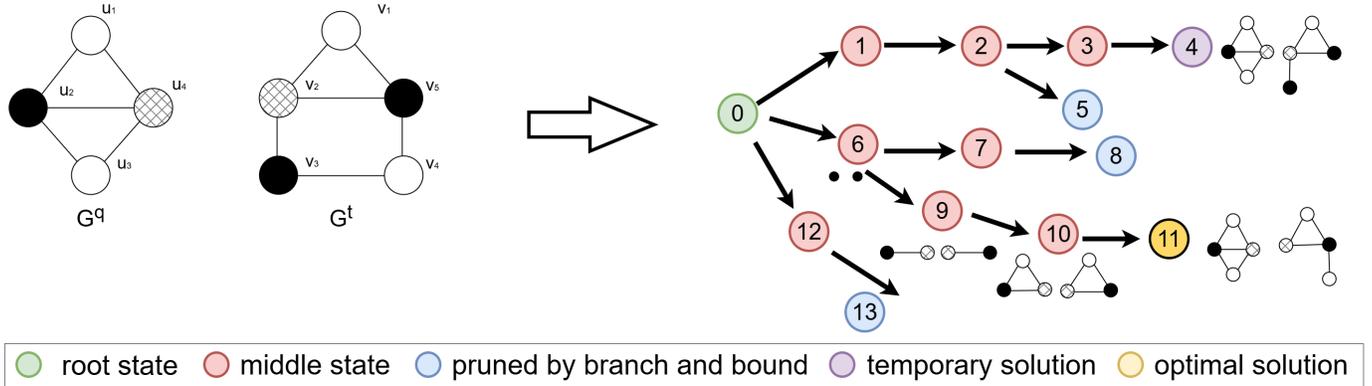


Fig. 2: An illustration of the search process of ASM on (G^q, G^t) . The branch-and-bound search algorithm (Algorithm 1) produces a tree structure, where each node represents a state (s_t) , the node ID reflects the order in which the state is visited, and each direct edge represents an action (a_t) , which adds a node-pair to the current node-node mapping M_t . The search is essentially depth-first with pruning through the lower bound check. The policy (Line 12 of Algorithm 1) refers to a node-pair selection strategy, i.e., which state to visit next? The visiting order affects the performance of searching in the tree. For example, if state 6 can be visited before state 1, a better solution can be found in fewer iterations. This means the GED of the current best match *currentMiniDist* will be smaller, allowing the algorithm to prune more branches in subsequent search steps. Hence, the search efficiency will be higher. When the search completes or a pre-defined search iteration budget is exhausted, the best solution identified by then will be returned. For the clarity of visualization, some nodes and edges in the search tree are omitted.

and the edge distance is

$$D_E(e, M(e)) = \begin{cases} 0 & e \in E^q, M(e) \in E^t \\ 1 & e \in E^q, M(e) \notin E^t \end{cases} \quad (3)$$

Here $\mathcal{A}(u)$ denotes the label of node u . Let e be the edge between nodes u_1 and u_2 from G^q , and $M(e)$ refers to the edge between $M(u_1)$ and $M(u_2)$ from G^t .

B. The Search Algorithm for ASM

Since the branch-and-bound search algorithm serves as the foundation of our proposed method, we first review a typical branch-and-bound search algorithm for ASM [12]. We then discuss its limitations and constraints, which lead to our proposed RL-ASM.

As shown in Algorithm 1 and Fig. 2, the branch-and-bound search algorithm, as proposed in [12], starts from an initial state with an empty mapping M_0 , and adds one pair of nodes to the mapping each time, while maintaining the best solution found so far. At each search step, denote the current search state as s_t , which consists of G^q , G^t and the current node-node mapping M_t . The algorithm attempts to select a node pair (u, v) , where $u \in G^q$ and $v \in G^t$, as action a_t , and add the pair to M_t . As shown in Fig. 2, each action (represented by a directed edge in the search tree) updates one state to another. For example, action (u_2, v_5) updates state 0 to state 6, starting a branch that includes states 7-11. In [12], the authors propose a method to calculate the lower bound of $C(M; G^q, G^t)$ for action selection. At Line 7 of Algorithm 1, the algorithm calculates the lower bound for each branch resulting by an action from action space A_t . For example, as shown in Fig. 2, when the current state is state 0, the algorithm calculates the lower bounds corresponding to the branches starting from states 1, 6, and 12. Then the algorithm compares

these lower bounds with *currentMiniDist* – the GED of the current best match. Any actions leading to the branches with the lower bounds greater than *currentMiniDist* will be eliminated from action space A_t (Line 8). If all the actions in A_t are eliminated, the algorithm will backtrack to the parent search state (Lines 9-11), i.e., the current branch will be cut off. When all of the nodes in G^q have been mapped, the algorithm compares its $C(M_t; G^q, G^t)$ with *currentMiniDist*. If $C(M_t; G^q, G^t)$ is smaller, indicating a better solution than current best mapping is identified, the algorithm sets the $C(M_t; G^q, G^t)$ as the new *currentMiniDist* and updates M_t to M^* (Lines 17-20). Note that if the method can find good matches early, *currentMiniDist* will be small, which means a lot of branches would be pruned and the search space could be reduced substantially.

The above method leverages a heuristic for action selection, denoted as “policy” at Line 12. Typically, a greedy policy is adopted, where the action leading to the branch with the minimum lower bound of GED is selected. A significant limitation of this method is that the lower bound based heuristics is not adaptive to the complex real-world graph structures because the method cannot fully exploit the information embedded in the graphs. More importantly, the greedy algorithm focuses on a locally optimal objective for action selection at the current step without considering potential better matches in the future steps, leading to sub-optimal solutions.

IV. PROPOSED METHOD

In this section, we introduce our Reinforcement Learning-based Approximate Subgraph Matching (RL-ASM). We provide a high-level overview of RL-ASM in Section IV-A, including the definitions of state, action, and reward. Section IV-B describes the details of our framework, focusing on

Algorithm 1 Branch-and-Bound for Approximate Subgraph Matching [12]

Input: query graph G^q and target graph G^t
Output: optimal node-node mapping M^*

- 1: Initialize $s_0 \leftarrow (G^q, G^t, M_0 = \emptyset)$
- 2: Initialize $stack \leftarrow \text{new Stack}(s_0)$
- 3: Initialize $currMiniDist \leftarrow \infty$
- 4: **while** $stack \neq \emptyset$ **do**
- 5: $s_t \leftarrow stack.pop()$
- 6: $A_t = s_t.get_actionspace()$
- 7: $Lbounds \leftarrow s_t.get_lowerbounds(A_t)$
- 8: $A_t \leftarrow \text{prune_actionspace}(A_t, Lbounds, currMiniDist)$
- 9: **if** $|A_t| = 0$ **then**
- 10: continue
- 11: **end if**
- 12: $a_t \leftarrow \text{policy}(s_t, A_t)$
- 13: $A_t \leftarrow A_t - \{a_t\}$
- 14: $M_t \leftarrow s_t.get_mapping()$
- 15: $M_t \leftarrow M_t + a_t$
- 16: **if** $|M_t| = |V^q|$ **then**
- 17: **if** $C(M_t; G^q, G^t) < currMiniDist$ **then**
- 18: $currMiniDist \leftarrow C(M_t; G^q, G^t)$
- 19: $M^* \leftarrow M_t$
- 20: **end if**
- 21: continue
- 22: **end if**
- 23: $stack.push(s_t)$
- 24: $s_{t+1} \leftarrow \text{environment.update}(s_t, a_t)$
- 25: $stack.push(s_{t+1})$
- 26: **end while**

the design of representation learning with Graph Transformer for ASM. This is followed by Sections IV-C and IV-D, where the model training is elaborated. In Section IV-E, we discuss why structural encoding features and Graph Transformer are required for ASM.

A. Overview

We formulate the ASM as a Markov Decision Process (MDP). The state s_t consists of two components: (1) query graph G^q and target graph G^t , and (2) the current node-node mapping M_t from V^q to V^t . Action $a_t : u \rightarrow v$ is defined as adding a node pair (u, v) to M_t , where $u \in V^q$ and $v \in V^t$. Our policy assigns a score to each action in action space A_t , and is modeled as a neural network $P_\theta(a_t|s_t)$, parameterized by θ , that computes a probability distribution over A_t given the current state s_t .

For subgraph matching, the immediate reward $r_t(s_t, a_t) = r_t^{\text{node}} + r_t^{\text{edge}}$ consists of two components: the node-matching reward r_t^{node} and the edge-matching reward r_t^{edge} . The node-matching reward r_t^{node} is determined by the label compatibility of the nodes added by $a_t : u \rightarrow v$, yielding +1 for identical labels and -1 otherwise. The edge-matching reward r_t^{edge} is defined as $|E_q^+| - |E_q^-|$. Here, let E_q be the existing edges between node u and all the mapped nodes of G^q at state s_t . E_q^+ consists of all edges $e \in E_q$ whose mapped edge $M_t(e)$ exists

in E^t . Conversely, E_q^- includes those edges $e \in E_q$ whose mapped edge does not exist in E^t . For instance, at state 10 of Fig. 2 with $M_{10} = \{(u_1, v_1), (u_2, v_5), (u_4, v_2)\}$ and $a_{10} : u_3 \rightarrow v_4$, $E_q^+ = \{(u_2, u_3)\}$ since (v_4, v_5) , the mapped edge, is in E^t , while $E_q^- = \{(u_3, u_4)\}$ since (v_2, v_4) , the mapped edge, is not in E^t . Thus, $r_{10}^{\text{edge}} = 0$. Essentially, $r_t(s_t, a_t)$ measures the change of GED after adding node pair (u, v) (a.k.a taking action a_t) to the current mapped subgraphs of query graph and target graph at state s_t .

If we do not impose any constraints to the action space, the cardinality of the action space will be $O(|V^q| \times |V^t|)$, which can be prohibitively huge. To reduce the action space size, we generate ϕ , the mapping order of nodes in G^q , before searching for subgraph matches. At each step, we take one node from G^q according to the order ϕ , and map this node to one of the nodes in G^t . We generate the order ϕ with the method proposed in [28], which is designed for the exact subgraph matching [29]. The algorithm starts by identifying the node $u \in V^q$ that possesses the maximum degree, and adds it to ϕ . In the subsequent step, the algorithm prefers to add those nodes that have a larger number of edges with the nodes already present in ϕ . This is an effective method to create ϕ because the node with the highest degree and the ones with more mapped neighbors typically encode substantial structural information, and thereby facilitate accurate pattern matching to initiate the search.

To enhance the efficiency in the tree search, we further design a mechanism that caches the lower bounds of branches and the policy scores of actions evaluated in previous steps. This allows our method to reuse cached results when backtracking to earlier states. Because of the strong correlation between the lower bounds and the GED of current best solution $currMiniDist$ [12], we delete the cached results whenever $currMiniDist$ is changed, and recalculate the lower bounds and the policy scores when backtracking to these states.

B. The RL-ASM Framework

As shown in Fig. 3, our RL-ASM framework consists of an encoder that produces the node embeddings for V^q and V^t , and a decoder that transforms the embeddings into a probabilistic distribution over actions $P_\theta(a_t|s_t)$ for action selection. To design this model, we face several challenges: (i) many existing graph neural networks (GNNs), which primarily follow a message passing paradigm, are inherently local and their expressiveness cannot exceed the 1-Weisfeiler-Lehman test [30]. Hence, these models lack the capability to tackle the complex approximate subgraph matching problem (see Section IV-E for justifications); (ii) as the task involves both query and target graphs, effectively sharing information between the two graphs presents a significant challenge; and (iii) each state s_t contains G^q , G^t , and the node mapping M_t ; it is challenging to effectively encapsulate the entire state's information into a representation that can be utilized to predict $P_\theta(a_t|s_t)$. These challenges have guided the design of our model.

1) *Node Features*: We employ three distinct features for each node: (1) **Label encoding**: A one-hot vector that specifies

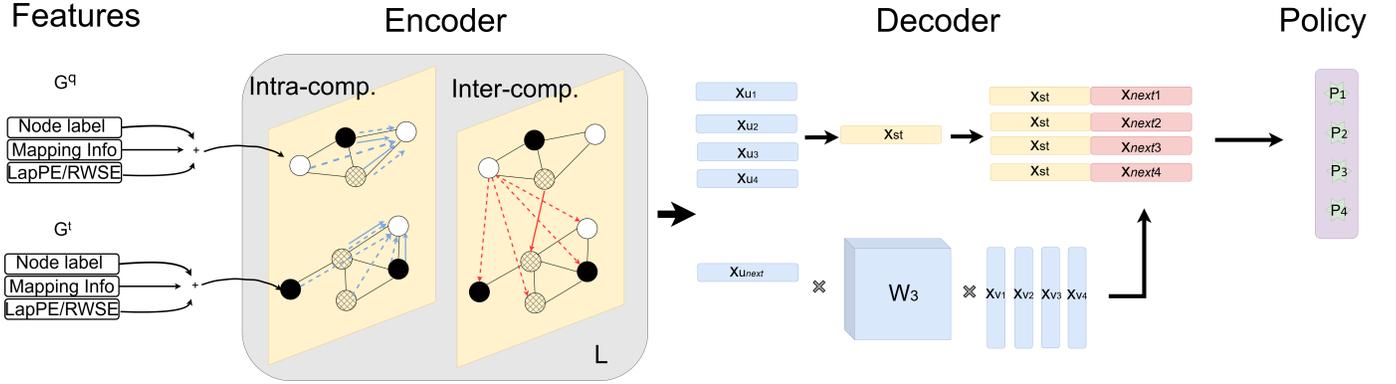


Fig. 3: Overview of RL-ASM. RL-ASM consists of two major components: encoder and decoder. The encoder processes node label, mapping info, positional and/or structural encodings by alternating intra- and inter-components L times (with L layers) to extract powerful node representations. The decoder leverages self-attention to node embeddings of G^q to generate a global state representation \mathbf{x}_{st} . The action representations are then generated by the product of embeddings of node $u_{next} \in G^q$ (according to ϕ), learnable weight tensors \mathbf{W}_3 , and embeddings of unmapped candidate nodes v_1, v_2, v_3 , and v_4 from G^t . Subsequently, the representations of state and actions are concatenated, which is fed to a MLP and a softmax classifier to calculate the probability distribution over the actions $P_\theta(a_t|s_t)$.

the label of each node; (2) **Mapping status**: A binary number indicating the node’s mapping status - 1 if mapped, and 0 otherwise; this feature is useful to alleviate Challenge (iii); (3) **Positional and structural encodings**: For node’s positional information, we utilize the Laplacian Positional Encoding (LapPE), which involves the eigenvectors associated with the k smallest non-zero eigenvalues of the graph Laplacian [31]. For structural information, we apply Random Walk Structural Encoding (RWSE), derived from the diagonal of the m -step random walk matrix [32]. Rampášek et al. [13] and Kreuzer et al. [31] have demonstrated theoretically and empirically that using positional encoding and structural encoding in GNNs can enhance the model’s expressiveness. Depending on the characteristics of the graph, we choose either positional or structural encoding or both as node features. This alleviates Challenge (i). Positional encoding is particularly effective for image graphs [33], while structural encoding is well suited for molecular graphs [34]. Each of the three features is processed by a distinct linear layer and subsequently concatenated as the feature representations, which are fed to the encoder.

2) *Encoder*: The encoder consists of an intra-component module and an inter-component module as shown in Fig. 3. The intra-component module processes and aggregates messages within query graph G^q and target graph G^t , separately. The inter-component module shares messages across the graphs based on the relationships of nodes in G^q and G^t .

To alleviate Challenge (i) and effectively utilize both local and global graph information, we employ the GraphGPS layer [13] as the intra-component. This hybrid layer combines a Message Passing Neural Network (MPNN) and a global attention mechanism, and offers greater expressiveness than traditional MPNNs, such as GCN [35], GAT [36], and GraphSAGE [30]. The MPNN propagates messages along the edges, whereas the global attention spreads information throughout the entire graph, as illustrated by the solid and dotted blue lines in Fig. 3. The global attention mechanism enhances this

capability by passing messages across all nodes, and therefore overcomes the expressiveness bottleneck associated with over-smoothing and over-squashing [13], [37]. In each GraphGPS layer, node features are updated by integrating outputs from both the MPNN and the global attention instances. Specifically, the GraphGPS layer is formulated as

$$\mathbf{X}_{\text{intra}}^{\ell+1} = \text{GPS}^\ell(\mathbf{X}^\ell, \mathbf{A}), \quad (4)$$

which is implemented as

$$\mathbf{X}_M^{\ell+1} = \text{MPNN}^\ell(\mathbf{X}^\ell, \mathbf{A}), \quad (5)$$

$$\mathbf{X}_H^{\ell+1} = \text{GlobalAttn}^\ell(\mathbf{X}^\ell), \quad (6)$$

$$\mathbf{X}_{\text{intra}}^{\ell+1} = \text{MLP}^\ell(\mathbf{X}_M^{\ell+1} + \mathbf{X}_H^{\ell+1}), \quad (7)$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of a graph of N nodes; $\mathbf{X}^\ell \in \mathbb{R}^{N \times d}$ denotes the d -dimensional node features for N nodes at the ℓ -th layer; MLP^ℓ is a 2-layer multilayer perceptron (MLP) block; and $\mathbf{X}_{\text{intra}}^{\ell+1} \in \mathbb{R}^{N \times d}$ represents the output of the intra-component at the ℓ -th layer. MPNN^ℓ and GlobalAttn^ℓ are modular components, which are implemented as GatedGCN [38] and Self-Attention [39], respectively.

As for the inter-component module of the encoder, we employ a cross-attention mechanism that enables the model to learn how to share messages between G^q and G^t . This mechanism is established based on the node mapping relationships. For state s_t , the current mapping M_t is $\{u \rightarrow v \mid u \in V_t^q, v \in V^t\}$, where V_t^q represents the set of nodes of G^q that have been mapped at step t . Our method allows the nodes of G^q to be mapped to nodes in G^t even if their labels differ. Consequently, for any node $u' \in G^q$ that remains unmapped, we define its candidate set $C_{u'}$ as $\{v' \in V^t \mid v' \text{ is unmapped in } G^t\}$. Therefore, the mapping describing the candidate relationships between nodes can be formulated as $M'_t = \{u' \rightarrow C_{u'}, \forall u' \in V_q \setminus V_t^q\}$. Let \widetilde{M}_t denote the union of M_t and M'_t , and \widetilde{M}_t^{-1} denote the reverse mapping of \widetilde{M}_t . The messages passing along M_t

and M'_t are shown as solid and dotted red lines in Fig. 3, respectively. Specifically, the message passing from G^q to G^t is implemented as follows:

$$\mathbf{X}_{\text{inter}}^\ell[u, :] = \mathbf{W}_1^\ell \mathbf{X}_{\text{intra}}^\ell[u, :] + \sum_{v \in \widetilde{M}_t(u)} \alpha_{uv} \mathbf{W}_2^\ell \mathbf{X}_{\text{intra}}^\ell[v, :], \quad (8)$$

where $\mathbf{X}_{\text{inter}}^\ell[u, :]$ and $\mathbf{X}_{\text{intra}}^\ell[u, :] \in \mathbb{R}^d$ represent inter- and intra-embeddings of node $u \in V_q$ at the ℓ -th layer, respectively; the attention coefficients α_{uv} are derived using multi-head dot-product attention [39], and \mathbf{W}_1^ℓ and \mathbf{W}_2^ℓ are the learnable matrices associated with the ℓ -th layer. Additionally, we implement the message passing from G^t to G^q following the reverse mapping \widetilde{M}_t^{-1} . The inter-component module is crucial for information exchange across the graphs, which addresses Challenge (ii).

The output from the inter-component module is then fed to an activation function, denoted as $\mathbf{X}^{\ell+1} = f_{\text{activate}}(\mathbf{X}_{\text{inter}}^{\ell+1})$, yielding the output of the ℓ -th encoder layer. We construct the encoder by stacking four such layers and employ jump knowledge [40] to aggregate the outputs from these layers.

3) *Decoder*: The node embeddings of query graph $\mathbf{X}^q \in \mathbb{R}^{|V^q| \times d}$, produced by the encoder, capture the information from G^q and from G^t through the cross-graph message passing manifested by \widetilde{M}_t and \widetilde{M}_t^{-1} . This allows \mathbf{X}^q to gather comprehensive information required to represent the entire state. To alleviate Challenge (iii), we introduce an attention-based mechanism that calculates the state embedding \mathbf{x}_{s_t} based on the embeddings of the query graph nodes as follows:

$$\mathbf{x}_{s_t} = \text{Pooling}(\text{SelfAttention}(\mathbf{X}^q)). \quad (9)$$

Here, we utilize a self-attention mechanism [41] on \mathbf{X}^q , followed by aggregation of the resulting representations using a pooling operation. In our experiments, average pooling is employed for aggregation.

Given that the policy is formulated as $P_\theta(a_t|s_t)$, it is essential to learn the representations of state s_t and action a_t . Consider action $a_t : u \rightarrow v$, with \mathbf{x}_u^1 and $\mathbf{x}_v \in \mathbb{R}^d$ denoting the node representations learned by the encoder, we employ a bilinear tensor product defined by a learnable tensor $\mathbf{W}_3 \in \mathbb{R}^{d \times d \times F}$ to ensure sufficient interaction between the node embeddings involved by action. Here, F is a hyperparameter that enhances the model's capacity by adding dimensions to learn complex relationships. Then action $a_t : u \rightarrow v$ can be expressed as $\mathbf{x}_u^T \mathbf{W}_3 \mathbf{x}_v$, which is concatenated with the state embedding \mathbf{x}_{s_t} . The combined vector is then fed into a multi-layer perceptron (MLP), followed by a softmax layer to generate

$$P_\theta(a_t|s_t) = \text{SoftMax}(\text{MLP}(\text{CONCAT}(\mathbf{x}_{s_t}, \mathbf{x}_u^T \mathbf{W}_3 \mathbf{x}_v))). \quad (10)$$

C. Policy Training

The goal of policy training is to maximize the accumulative long-term reward: $R_t = \sum_{i=t}^T \gamma^{i-t} r_i$, where T is the total number of time steps of an episode, γ is the discount factor,

¹Here, u is the next node selected from G^q according to the order ϕ .

and r_i is the immediate reward incurred by action a_i at the i -th step. To enhance training efficiency, we disable the backtracking mechanism during the policy training. That is, once our algorithm reaches to a leaf node in the search tree, the episode ends. We employ the Proximal Policy Optimization (PPO) [42], one of the most effective policy gradient methods, to train our model. PPO mitigates excessively large policy changes by clipping policy probability ratios, which helps maintain stability in the learning process. The core term of the PPO loss function is defined as

$$L^{\text{clip}}(\theta) = -\hat{\mathbb{E}}_t \left[\min \left(\rho(\theta) \hat{R}_t, \text{clip}(\rho(\theta), 1-\varepsilon, 1+\varepsilon) \hat{R}_t \right) \right], \quad (11)$$

where $\hat{\mathbb{E}}_t[\cdot]$ indicates the empirical average over a batch of samples, $\rho(\theta) = \frac{P_\theta(a_t|s_t)}{P_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of the action probabilities under current and old policy, and \hat{R}_t denotes the accumulated reward, normalized across the batch. The operation $\text{clip}(\rho(\theta), 1-\varepsilon, 1+\varepsilon) \hat{R}_t$ modifies the objective by clipping the probability ratio, thereby moderating the influence of extreme policy changes during the training. To ensure sufficient exploration and prevent premature convergence to a local optimal policy, we also incorporate an entropy bonus term to the loss function:

$$L^{\text{entrop}}(\theta) = -\hat{\mathbb{E}}_t [H(P_\theta(\cdot|s_t))], \quad (12)$$

where $H(\cdot)$ is the entropy of the probability distribution over action space A_t given state s_t . The overall objective function of PPO is

$$L^{\text{PPO}}(\theta) = L^{\text{clip}}(\theta) + cL^{\text{entrop}}(\theta), \quad (13)$$

where c is a hyperparameter that balances the contributions from the two loss terms.

D. Pre-training

For graph pairs of large graphs, the action space can be huge. To expedite the learning process and enhance sample efficiency, we pre-train our model with imitation learning before the PPO fine-tuning. This pre-training involves sampling subgraphs from the target graph and recording the correspondence between nodes of the sampled and target graphs to define expert actions. Noise is added to these sampled graphs to generate query graphs. In this pre-training stage, the model's predictions are compared to the expert's choices, with training conducted using a supervised cross-entropy loss:

$$L^{\text{imit}}(\theta) = - \sum_{a \in A_t} y_a \log(\hat{y}_a(\theta)), \quad (14)$$

where y_a indicates whether the action a aligns with the expert's choice (1 for yes, 0 for no), and \hat{y}_a represents the predicted probability corresponding to action a . Note that during imitation learning, we enable Dropout and Batch Normalization in the model, whereas in the PPO fine-tuning, we disable these components to improve the training stability.

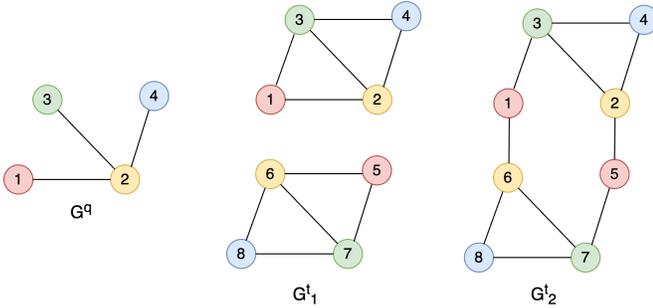


Fig. 4: Example graphs that are indistinguishable by MPNN.

E. Why are LapPE and Graph Transformer Needed for ASM?

The encoder of RL-ASM consists of an intra-component module and an inter-component module. In the inter-component, messages are passed along \widetilde{M} and \widetilde{M}^{-1} , which denote the edges between the nodes in G^q and G^t . This implies that the inter-component can be considered as an MPNN. Therefore, if we employ traditional MPNN architectures (e.g., GCN, GAT, or GatedGCN) as the intra-component of our encoder, the entire architecture will be an MPNN. It has been proven that the expressiveness of MPNNs cannot exceed the 1-Weisfeiler-Lehman test [30]. That is, they cannot differentiate non-isomorphic graphs by using only 1-hop neighborhood aggregation. In Fig. 4, when only the labels of nodes (which correspond to node colors in the figure) are used as input features, the MPNN fails to distinguish between G_1^t and G_2^t because the nodes with the identical ID in G_1^t and G_2^t have the same 1-hop neighborhood structure. Consequently, when attempting to match query graph G^q with G_1^t and G_2^t , an MPNN-based model will yield identical matching results. However, the optimal matching solutions for these cases are different; one is $\{(1,1), (2,2), (3,3), (4,4)\}$ and another one is $\{(1,5), (2,2), (3,3), (4,4)\}$. This example illustrates the expressiveness limitation of MPNNs when addressing the ASM problem.

Kreuzer et al. [31] demonstrate that with the full set of Laplacian eigenvectors, a Graph Transformer model can be a universal function approximator on graphs and can provide an approximate solution to the graph isomorphism problem. This suggests that by integrating LapPE and GraphGPS (a type of Graph Transformer) into the model, our RL-ASM has the capability beyond the traditional MPNN models in solving the ASM problem.

V. EXPERIMENTS

In this section, we compare our RL-ASM with state-of-the-art approaches for ASM, and demonstrate its effectiveness and efficiency. All our experiments were conducted using a server equipped with Intel Xeon Gold 6242 CPUs and NVIDIA Tesla V100 GPUs. For reproducible research, we provide our source code at <https://github.com/KaiyangLi1992/RL-ASM>.

A. Datasets

Our experiments utilize one synthetic dataset and three real-world datasets from diverse domains: biology, computer

Dataset	SYNTHETIC	AIDS	MSRC_21	Email
Target size (nodes)	100	17.03	77.48	1005
Target size (edges)	196	17.58	198.18	25571
Query size (nodes)	15.75	8.09	24.12	13.19
Query size (edges)	18.13	8.24	57.29	60.04
Num. of Node Labels	13	40	27	47
Sampling range of $ V^q $	[10, 20]	[6, 10]	[16, 32]	[8, 16]

TABLE I: Statistics of query graphs and target graphs.

vision, and social networks. These datasets are described as follows:

- **SYNTHETIC** [43]: This dataset comprises 300 synthetic graphs generated by a statistical model. Each graph consists of 100 nodes and 196 edges, and each node is endowed with a normally distributed scalar label.
- **AIDS** [43]: This dataset contains 2,000 graphs, each representing a molecular compound. Nodes in these graphs correspond to atoms, while edges represent the chemical bonds between them.
- **MSRC_21** [43]: As a semantic image processing benchmark, this dataset includes 563 graphs, each representing the graphical model of an image. Nodes in the graph represent objects in an image, and edges illustrate the relationships among the objects.
- **EMAIL** [44]: Originating from an email communication network at a European research institution, this dataset uses nodes to represent employees, and edges to represent the email interactions between employees. The node label indicates the department to which each employee belongs.

For datasets **SYNTHETIC**, **AIDS**, and **MSRC_21**, which contain multiple graphs, we randomly select graphs from these collections to serve as the target graphs. For the **EMAIL** dataset, which contains a single graph, we use this graph consistently as the target graph. We employ the Breadth-First Search (BFS) on these target graphs to sample nodes, ceasing the sampling when the node count reaches the parameters specified in Table I. From these sampled nodes, induced sub-graphs are generated to form the *seed* query graphs. We then introduce noise to these *seed* query graphs by adding edges and altering node labels, thereby producing query graphs with noise levels of 0%, 5%, and 10%. Specifically, we calculate the total number of nodes and edges in each query graph, multiply this total number by the noise level to determine the noise intensity, N_{noise} . We then randomly select an integer N_{noise}^{node} between 0 and N_{noise} , change the labels of N_{noise}^{node} nodes, and add $N_{noise} - N_{noise}^{node}$ edges to the original query graph, thereby generating a noisy query graph. Each pair of a query graph and its corresponding original target graph constitutes a graph pair for ASM. After eliminating duplicate graph pairs, we partition the dataset into training, validation, and test sets following an 8:1:1 ratio. The statistics of the query and target graphs are reported in Table I.

To closely simulate real-world conditions, we operate under the assumption that the noise level in query graphs remains unknown. To ensure our model adapts broadly, our training and validation sets for each dataset encompass query graphs from all three noise levels: 0%, 5%, and 10%. However, for the purpose of evaluation, we specifically test our model’s performances on these noise levels to report its performances

Training Stage	Hyperparameter	AIDS	SYNTHETIC	MSRC_21	EMAIL
	Hidden Dimension	32	32	32	32
	# Encoder Layers	4	4	4	4
	Interaction Dimension F	32	32	32	32
	Positional and Structural Encodings	RWSE	RWSE	LapPE	RWSE
	Batch Size	1024	1024	1024	1024
	Learning Rate	0.001	0.0005	0.001	0.001
Imitation Learning	#Epochs	1000	1000	1000	1000
	Weight Decay	0.01	0.01	0.01	0.01
	Learning Rate	0.005	0.001	0.0005	0.0005
PPO Fine-tuning	Entropy Coefficient	0.01	0.01	0.01	0.01
	Batch Size	2048	512	1024	64
	# Epochs	10	10	10	10
	γ	0.99	0.99	0.99	0.99
	Clipping Range	0.2	0.2	0.2	0.2

TABLE II: The hyperparameters of RL-ASM used for different benchmark datasets.

under different conditions.

B. Baselines

We compare our method with a neural-network based method *NeuroMatch* [45] and two heuristic methods *ISM* [12] and *APM* [10]. As shown in Fig. 2, a good initial matching solution is crucial for the search performance of branch-and-bound methods. Therefore, we present the first-round matching results (i.e., the results without backtracking) for both *ISM* and our method in this section. Additionally, we emphasize the role of imitation learning as an important pre-training stage of our method, showcasing the performances of our model when trained exclusively through imitation learning. We benchmark against the following methods:

- **NeuroMatch** [45] decomposes query and target graphs into small subgraphs and embeds them using graph neural networks. It first predicts the probability of nodes in query graphs matching to nodes in target graphs with the embeddings of the decomposed subgraphs. It then utilizes the Hungarian algorithm [46] to map the nodes between the two graphs such that the sum of the probabilities of the matched node pairs is maximized with the constraint that each node in query graph is assigned to a different node in target graph.
- **APM** [10] reduces ASM to exact subgraph matching. It focuses on identifying exact matches between prototype graphs (i.e., those derived from the query graph with GED below a specific threshold) and target graphs. To avoid an excessively large search space, the method only considers prototypes that are derived by adding edges to query graphs, excluding those that are derived by altering nodes' labels.
- **ISM** [12] formulates ASM as a tree search problem. It calculates the lower bound of the GED between query graph and the corresponding subgraph in target graph within each search branch from the current step. The branches, whose lower bounds surpass the GED of the best match found so far, are pruned. The method employs a greedy strategy to select the node pair that leads to the branch with the minimal lower bound at each search step.
- **RL-ASM-IL** showcases the first-round mapping results of our RL-ASM, which is trained exclusively through imitation learning. This illustrates the impact of imitation learning to our model.

- **RL-ASM-FR** and **ISM-FR** illustrate the first-round matching results of RL-ASM and ISM without backtracking.

For the experiments of *ISM* and *RL-ASM*, we limit the search tree exploration to 600 seconds. If the programs fail to traverse the entire tree within this time limit, we terminate the execution and treat the best matching result found so far as the final outcome.

The hyperparameters of *RL-ASM* used for different benchmark datasets are provided in Table II. We select the best model according to the metric on the validation set and report the metric on the test set. Specifically, the learning rates for imitation learning and PPO fine-tuning are selected from $\{5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2\}$. We search for the hidden dimension of nodes from $\{16, 32, 64\}$ for *RL-ASM*.

C. Results

1) *Effectiveness of Approximate Subgraph Matching*: The graph edit distance (GED), as reported in Table III, quantifies the discrepancies between the query graph and its corresponding subgraph in the target graph, with smaller values indicating better matches. The results of *RL-ASM-IL* indicate that even when trained solely via imitation learning, our model's initial matching results significantly outperform the heuristic methods: *APM* and *ISM*. This demonstrates our model's ability to effectively learn graph information via Graph Transformer for ASM. Compared with *NeuroMatch* and *RL-ASM-IL*, *RL-ASM-FR* shows superior performance on the **SYNTHETIC**, **MSRC_21**, and **EMAIL** datasets. This improvement attributes to the RL training phase, which maximizes an accumulative long-term reward. The enhancements observed from RL training on the **MSRC_21** and **EMAIL** datasets are more pronounced compared to the **AIDS** dataset. This is because the graphs in **MSRC_21** and **EMAIL** are much larger and more challenging, requiring the model to engage in optimizing long-term rewards and explore a broader solution space, the areas where RL excels. *RL-ASM* delivers the best results by effectively searching the trees with backtracking to find better subgraph matches beyond the results of *RL-ASM-FR*. In the following subsection, we will demonstrate that our model, guided by a neural network, can potentially identify optimal solutions within a constrained time limit.

When no noise is added to query graphs (e.g., noise ratio 0%), *APM* can find the optimal solution because it reduces the

TABLE III: Graph Edit Distance (GED) between query graph and its matched subgraph identified from target graph. Note that the ‘‘GroundTruth’’ number denotes the GED between the ‘‘noisy’’ query graph and its corresponding *seed* query graph sampled from target graph.

Dataset	SYNTHETIC			AIDS			MSRC_21			EMAIL			
	Noise Ratio	0%	5%	10%	0%	5%	10%	0%	5%	10%	0%	5%	10%
GroundTruth		0.00	1.70	3.39	0.00	1.00	1.67	0.00	3.99	8.03	0.00	3.66	7.32
NeuroMatch [45]		1.52	4.42	7.96	0.80	3.12	7.49	3.56	11.46	22.43	3.71	17.41	26.51
APM [10]		0.00	2.45	4.57	0.00	1.24	2.43	0.00	7.46	13.43	0.00	8.11	15.61
ISM-FR		8.14	9.13	11.03	0.79	1.91	2.92	18.50	21.66	26.87	4.97	11.25	13.32
ISM [12]		7.00	8.35	10.60	0.01	1.04	2.00	17.07	20.69	26.26	4.67	10.74	12.90
RL-ASM-IL		0.00	1.65	3.40	0.06	1.16	1.85	0.01	4.29	8.93	0.00	3.65	7.00
RL-ASM-FR		0.00	1.59	3.31	0.06	1.17	1.84	0.00	4.19	8.93	0.00	3.56	6.63
RL-ASM (ours)		0.00	1.54	3.18	0.00	1.04	1.77	0.00	4.07	8.77	0.00	3.13	6.09

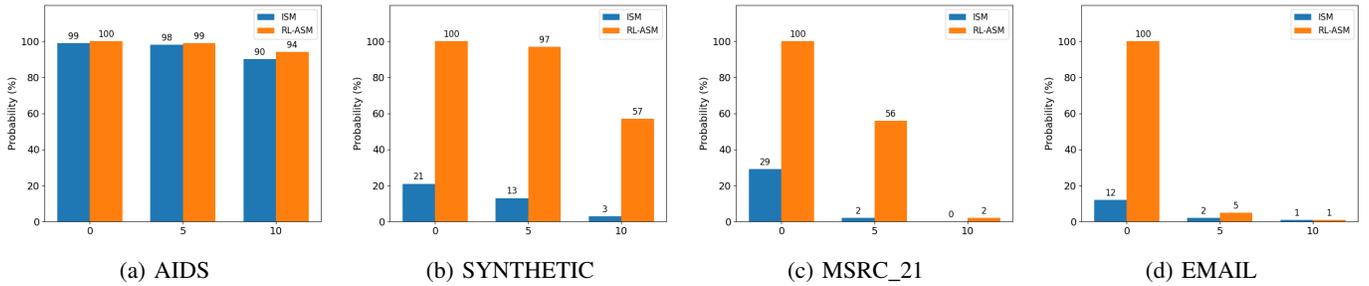


Fig. 5: The probabilities that ISM and our method find the optimal solutions within 600s.

ASM to an exact subgraph matching problem. For each graph pair, it attempts to match G^q and G^t using an exact subgraph mapping method, which can consistently identify the optimal solution when query graph G^q contains no noise. However, the performance of APM declines when handling noisy instances of G^q as it does not allow mappings between node pairs with differing labels.

For ISM, which selects actions based on heuristics, it struggles to find effective solutions in the initial round, as manifested by the results of ISM-FR in Table III. This limitation hinders its ability to prune numerous branches during tree searches. Moreover, the greedy search utilizes heuristics to make decision at each step. Consequently, both its initial and final mappings are prone to sub-optimal. In addition, NeuroMatch is designed for exact subgraph matching; when noise is added to the query graph, it struggles to map nodes accurately based on the learned representations.

2) *Efficiency of Approximate Subgraph Matching:* Both ISM and RL-ASM utilize a branch-and-bound framework to effectively search the entire tree to find the optimal solution. Fig. 5 illustrates the efficiencies of ISM and our method by reporting the probability of each method finding the optimal solution within 600 seconds. The results reveal that our method, guided by a neural network for action selection, is significantly more efficient than ISM. While both methods perform similarly with smaller graphs, such as the instances in the **AIDS** dataset, the efficiency gap is more pronounced with larger graphs. For example, on the **SYNTHETIC** dataset, our method significantly outperforms ISM, achieving a probability of identifying the optimal solution that is about 5x higher than

TABLE IV: Ablation study of design choices on node-sorting and GraphGPS.

Dataset	SYNTHETIC			
	Noise-Ratio	0%	5%	10%
RL-ASM w/o node-sorting		0	1.63	3.38
RL-ASM w/o GraphGPS		0	1.67	3.47
RL-ASM		0	1.54	3.18

that of ISM.

3) *Ablation Study:* In Section IV, we introduce a method based on [28] to sort the nodes in G^q for mapping order ϕ , and we also utilize GraphGPS [13], a type of graph transformer, as the intra-component in our encoder. This section assesses the impact of these design choices by replacing the sorted node order with a random node order or substituting GraphGPS with GatedGCN, which involves removing the global attention from the intra-component of our encoder. Table IV presents the results of these ablation studies. It is observed that both node sorting ϕ and GraphGPS enhance the performance of our RL-ASM. The node sorting prioritizes nodes with higher degrees and more matched neighbors, leveraging the structural information to facilitate accurate pattern matching. Moreover, GraphGPS, integrating an MPNN with global attention, significantly increases the expressiveness of our model.

4) *Generalization:* In the experiments above, we utilize the BFS to sample subgraphs and produce induced subgraphs (a.k.a query graphs) from the sampled nodes. In this section, we adopt a Random Walk (RW) approach for query graph sampling. Specifically, we start at a randomly selected node and move to an adjacent node with random walk, repeating

TABLE V: Effectiveness on the random walk sampled dataset.

Dataset	SYNTHETIC			
	Noise-Ratio	0%	5%	10%
ASP		0	2.03	4.04
ISM		5.58	7.69	8.51
RL-ASM		0	1.47	3.42

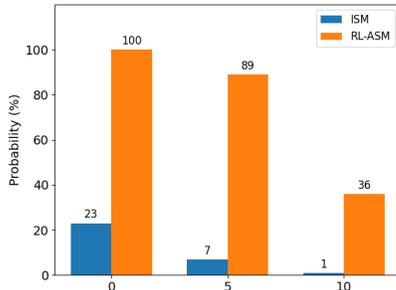


Fig. 6: Efficiency on the random walk sampled dataset.

this process for a pre-determined number of steps. During this process, we record all visited nodes and edges to construct the query graphs. Note that the RW sampled subgraphs may not necessarily be the induced subgraphs. Subsequently, we introduce noise to these sampled graphs to create noisy query graphs. This methodology is applied to the **SYNTHETIC** dataset to create training, validation, and test sets, which are used to assess the performance of ASM involving non-induced subgraphs. The GED between G^q and the matched subgraphs, along with the probability of identifying the optimal solutions within 600 seconds, are reported in Table V and Fig. 6, respectively. The results indicate that our model still outperforms the baseline methods in terms of effectiveness and efficiency, even when the query graphs are not induced ones, demonstrating the generalization of RL-ASM.

VI. CONCLUSION

As an NP-hard problem, approximate subgraph matching (ASM) is a fundamental research task in the database and graph mining communities. One significant strand of ASM methods follows the branch-and-bound search framework, where search performance is heavily influenced by the selection of actions (i.e., the mapped node pairs) in the search tree. We observe that existing methods, which utilize heuristics to select actions, lack the capability to fully exploit the structural and label information of graphs, and make sub-optimal decisions for ASM. In this paper, we introduce RL-ASM, a reinforcement learning based method for ASM that leverages Graph Transformer to extract the full graph information and optimizes an accumulative long-term rewards over episodes for ASM. Extensive experiments demonstrate the effectiveness and efficiency of our approach on four benchmark datasets. ASM is a relatively less investigated research area. We open source our code to facilitate the research in this area.

REFERENCES

[1] C. C. Aggarwal, H. Wang *et al.*, *Managing and mining graph data*. Springer, 2010, vol. 40.

[2] D. Chakrabarti and C. Faloutsos, *Graph mining: laws, tools, and case studies*. Springer Nature, 2022.

[3] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, “gstore: answering sparql queries via subgraph matching,” *Proc. VLDB Endow.*, vol. 4, no. 8, p. 482–493, may 2011. [Online]. Available: <https://doi.org/10.14778/2002974.2002976>

[4] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao, “Answering natural language questions by subgraph matching over knowledge graphs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 5, pp. 824–837, 2018.

[5] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel, “Saga: a subgraph matching tool for biological graphs,” *Bioinformatics*, vol. 23, no. 2, pp. 232–239, 2007.

[6] W. Fan, “Graph pattern matching revised for social network analysis,” in *Proceedings of the 15th International Conference on Database Theory*, ser. ICDT ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 8–21. [Online]. Available: <https://doi.org/10.1145/2274576.2274578>

[7] G. Lu, K. Li, X. Wang, Z. Liu, Z. Cai, and W. Li, “Neural-based inexact graph de-anonymization,” *High-Confidence Computing*, vol. 4, no. 1, p. 100186, 2024.

[8] M. Zaslavskiy, F. Bach, and J.-P. Vert, “Global alignment of protein-protein interaction networks by graph matching methods,” *Bioinformatics*, vol. 25, no. 12, pp. i259–i267, 2009.

[9] S. Ji, P. Mittal, and R. Beyah, “Graph data anonymization, de-anonymization attacks, and de-anonymizability quantification: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 1305–1326, 2016.

[10] T. Reza, M. Ripeanu, G. Sanders, and R. Pearce, “Approximate pattern matching in massive graphs with precision and recall guarantees,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1115–1131.

[11] S. Zhang, J. Yang, and W. Jin, “Sapper: Subgraph indexing and approximate matching in large graphs,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1185–1194, 2010.

[12] T. K. Tu, J. D. Moorman, D. Yang, Q. Chen, and A. L. Bertozzi, “Inexact attributed subgraph matching,” in *2020 IEEE international conference on big data (big data)*. IEEE, 2020, pp. 2575–2582.

[13] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, “Recipe for a general, powerful, scalable graph transformer,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 501–14 515, 2022.

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

[15] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, “Fast best-effort pattern matching in large attributed graphs,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 737–746.

[16] Y. Yuan, G. Wang, J. Y. Xu, and L. Chen, “Efficient distributed subgraph similarity matching,” *The VLDB Journal*, vol. 24, no. 3, pp. 369–394, 2015.

[17] S. Dutta, P. Nayek, and A. Bhattacharya, “Neighbor-aware search for approximate labeled graph matching using the chi-square statistics,” in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1281–1290.

[18] S. Agarwal, S. Dutta, and A. Bhattacharya, “Venom: Approximate subgraph matching with enhanced neighbourhood structural information,” in *Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD)*, 2024, pp. 18–26.

[19] —, “Chisel: Graph similarity search using chi-squared statistics in large probabilistic graphs,” *Proceedings of the VLDB Endowment*, vol. 13, no. 10, pp. 1654–1668, 2020.

[20] D. L. Sussman, Y. Park, C. E. Priebe, and V. Lyzinski, “Matched filters for noisy induced subgraph detection,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 11, pp. 2887–2900, 2019.

[21] E. Khalil, H. Dai, Y. Zhang, B. Dilikina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” *Advances in neural information processing systems*, vol. 30, 2017.

[22] C. Fan, L. Zeng, Y. Sun, and Y.-Y. Liu, “Finding key players in complex networks through deep reinforcement learning,” *Nature machine intelligence*, vol. 2, no. 6, pp. 317–324, 2020.

[23] Y. Liu, C.-M. Li, H. Jiang, and K. He, “A learning based branch and bound for maximum common subgraph related problems,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, 2020, pp. 2392–2399.

- [24] Y. Bai, D. Xu, Y. Sun, and W. Wang, “Gsearch: Maximum common subgraph detection via learning to search,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 588–598.
- [25] H. Wang, Y. Zhang, L. Qin, W. Wang, W. Zhang, and X. Lin, “Reinforcement learning based query vertex ordering model for subgraph matching,” in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 245–258.
- [26] Y. Bai, D. Xu, Y. Sun, and W. Wang, “Detecting small query graphs in a large graph via neural subgraph search,” *arXiv preprint arXiv:2207.10305*, 2022.
- [27] H. He and A. K. Singh, “Closure-tree: An index structure for graph queries,” in *22nd International Conference on Data Engineering (ICDE’06)*. IEEE, 2006, pp. 38–38.
- [28] 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, pp. 1–13, 2013.
- [29] 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.
- [30] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=ryGs6iA5Km>
- [31] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou, “Rethinking graph transformers with spectral attention,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 21 618–21 629. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/b4fd1d2cb085390fbbadae65e07876a7-Paper.pdf
- [32] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, “Graph neural networks with learnable structural and positional representations,” *arXiv preprint arXiv:2110.07875*, 2021.
- [33] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, “Vision gnn: An image is worth graph of nodes,” *Advances in neural information processing systems*, vol. 35, pp. 8291–8303, 2022.
- [34] R. Sun, H. Dai, and A. W. Yu, “Does gnn pretraining help molecular representation?” *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 096–12 109, 2022.
- [35] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [36] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [37] S. Akansha, “Over-squashing in graph neural networks: A comprehensive survey,” *arXiv preprint arXiv:2308.15568*, 2023.
- [38] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, “Gated graph sequence neural networks,” in *Proceedings of ICLR’16*, 2016.
- [39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [40] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” in *International conference on machine learning*. PMLR, 2018, pp. 5453–5462.
- [41] P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018, pp. 464–468.
- [42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [43] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, “Tudataset: A collection of benchmark datasets for learning with graphs,” in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. [Online]. Available: www.graphlearning.io
- [44] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graph evolution: Densification and shrinking diameters,” *ACM transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 2–es, 2007.
- [45] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec *et al.*, “Neural subgraph matching,” *arXiv preprint arXiv:2007.03092*, 2020.
- [46] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.