RLMCS: A REINFORCEMENT LEARNING APPROACH TO MAXIMUM COMMON SUBGRAPH DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Detecting the Maximum Common Subgraph (MCS) between two input graphs is fundamental for applications in biomedical analysis, malware detection, cloud computing, etc. This is especially important in the task of drug design, where the successful extraction of common substructures in compounds can reduce the number of experiments needed to be conducted by humans. However, MCS computation is NP-hard, and state-of-the-art exact MCS solvers do not have worst-case time complexity guarantee and cannot handle large graphs in practice. Designing learning based models to find the MCS between two graphs in an approximate yet accurate way while utilizing as few labeled MCS instances as possible remains to be a challenging task. Here we propose RLMCS, a Graph Neural Network based model for MCS detection through reinforcement learning. Our model uses an exploration tree to extract subgraphs in two graphs one node pair at a time, and is trained to optimize subgraph extraction rewards via Deep Q-Networks. A novel graph embedding method is proposed to generate state representations for nodes and extracted subgraphs jointly at each step. Experiments on real graph datasets demonstrate that our model performs favorably to exact MCS solvers and supervised neural graph matching network models in terms of accuracy and efficiency.

1 INTRODUCTION

Due to the flexible and expressive nature of graphs, designing machine learning approaches to solve graph tasks is gaining increasing attention from researchers. Among various graph tasks such as link prediction Zhang & Chen, graph classification Ying et al. (2018) and generation You et al., detecting the largest subgraph that is commonly present in both input graphs, known as Maximum Common Subgraph (MCS) (Bunke & Shearer, 1998) (as shown in Figure **??**), is relatively novel and less explored.

MCS naturally encodes the degree of similarity between two graphs, is domain-agnostic, and has various versions of definitions Duesbury et al. (2018), and thus has applications in many domains such as software analysis Park et al. (2013), graph database systems Yan et al. (2005) and cloud computing platforms Cao et al. (2011). In drug design, the manual testing of the effects of a new drug is known to be a major bottleneck, and the identification of compounds that share common or similar subgraphs which tend to have similar properties can effectively reduce the manual labor Ehrlich & Rarey (2011).

The main challenge in MCS detection is its NP-hard nature, causing the state-of-the-art exact MCS detection algorithms to run in exponential time in worst cases McCreesh et al. (2017); Hoffmann et al. (2017) and very hard to scale to large graphs in practice. The usefulness of MCS detection yet the inefficiency of exact MCS solvers call for the design of learning based approximate solvers, not only due to their leaning ability, potentially yielding good accuracy, but also because of the efficient nature of many such methods such as deep learning models.

However, existing machine learning approaches to graph matching either do not address the MCS detection task directly or rely on labeled data requiring the pre-computation of MCS results by running exact solvers. For example, there is large amount of works addressing image matching which turn images into graphs whose matching results have semantic meanings and thus do not satisfy the general-domain MCS constraints Zanfir & Sminchisescu (2018); Wang et al. (2019); Yu et al.

(2020). The graph alignment/matching task aims to find the node-node correspondence between two graphs yet the result is unrelated to MCS Xu et al. (2019b;a). Graph similarity computation Bai et al. (2019a); Li et al. (2019); Bai et al. (2020a) is more closely related, but only predicts a scalar score for two graphs instead of the node-node correspondonce.

To the best of our knowledge, the only existing model that addresses MCS directly is NEURALMCS Bai et al. (2020b). Although performing MCS detection in an efficient way, NEURALMCS must be trained in a completely supervised fashion which may potentially overfit and requires a large amount of labeled MCS instances. In practice, when labeled MCS results are scarce, how to design a machine learning approach that efficiently and effectively extracts the MCS remains a challenge.

In this paper, we present RLMCS, a general framework for MCS detection suited both when training pairs exist and when training pairs are unavailable. The model utilizes a novel **J**oint **S**ubgraph-**N**ode **E**mbedding (**JSNE**) network to perform graph representation learning, a Deep Q-Network (DQN) Mnih et al. (2015) to predict action distributions, and a novel exploration tree based on beam search to perform subgraph extraction iteratively. The entire model is trained end-to-end in the reinforcement learning (RL) framework. Besides, an approximate graph isomorphism checking algorithm is proposed specifically for the iterative MCS extraction procedure, named as **F**ast **I**terative **G**raph **I**somorphism (**FIGI**).

Experiments on synthetic and real graph datasets demonstrate that the proposed model significantly outperforms state-of-the-art exact MCS detection algorithms in terms of efficiency and exhibits competitive accuracy over other learning based graph matching models.

2 PRELIMINARIES

2.1 PROBLEM DEFINITION

We denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ denote the vertex and edge set. An induced subgraph is defined as $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ where $\mathcal{E}_s$ preserves all the edges between nodes in $\mathcal{V}_s$, i.e. $\forall i, j \in \mathcal{V}_s$, $(i, j) \in \mathcal{E}_s$ if and only if $(i, j) \in \mathcal{E}$. For example, in Figure 1, the five-member ring is an induced subgraph of $\mathcal{G}_1$ and $\mathcal{G}_2$ because all the five edges between the five nodes are included in the subgraph.

In this paper, we aim at detecting the Maximum Common induced Subgraph (MCS) between an input graph pair, denoted as $\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)$, which is the largest induced subgraph that is contained in both $\mathcal{G}_1$ and $\mathcal{G}_2$. In addition, we require $\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)$ to be a connected subgraph. We allow the nodes of input graphs to be labeled, in which case the labels of nodes in the MCS must match, as shown in Figure 1.

Lemma 2.1. For a given input graph pair $(\mathcal{G}_1, \mathcal{G}_2)$, the number of node in their MCS is bounded by the smaller of the two graphs, $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| \leq \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$.

Proof. Suppose $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| > \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$. However, by definition MCS is a subgraph that is contained in $\mathcal{G}_1$ and $\mathcal{G}_2$, which cannot be larger than either $|\mathcal{V}_1|$ or $|\mathcal{V}_2|$. By contradiction, $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| \leq \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$. □

$|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| = \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$ when $\mathcal{G}_1$ is subgraph isomorphic to $\mathcal{G}_2$ or $\mathcal{G}_2$ is subgraph isomorphic to $\mathcal{G}_1$. The task of subgraph isomorphism (checking if one graph is contained in another graph) can be regarded as a special case of MCS detection.

2.2 SEARCH ALGORITHM FOR MCS

Among various algorithms for MCS, we adopt the state-of-the-art search-based algorithm in our framework. The basic version, MCSPLIT, is presented in McCreesh et al. (2017) and the more recent version, MCSPLIT+RL, is proposed in Liu et al. (2019). The whole search algorithm, outlined in Algorithm 1, is essentially a classic branch-and-bound search algorithm that maintains a best solution found so far throughout the search with specific upper bound computation heuristic designed for the MCS task.

At each search state, denote the current solution as $\mathcal{G}_{1s} = (\mathcal{V}_{1s}, \mathcal{E}_{1s})$ and $\mathcal{G}_{2s} = (\mathcal{V}_{2s}, \mathcal{E}_{2s})$ along with the node-node mappings. A bidomain consists of two sets of nodes that are not in the current solution, $\langle \mathcal{V}_1', \mathcal{V}_2' \rangle$ where nodes in $\mathcal{V}_1'$ and $\mathcal{V}_2'$ have the same connectivity pattern with respect to the

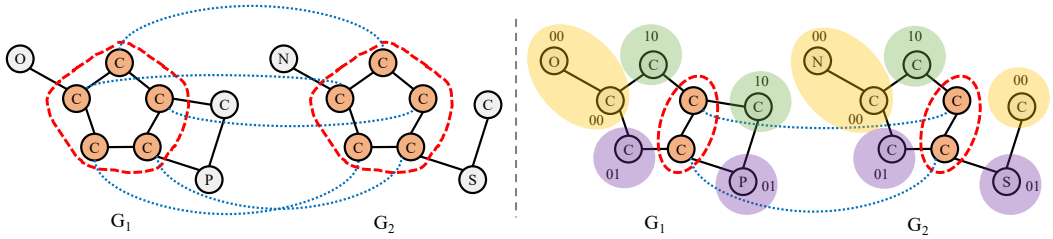

Figure 1: Left: For graph pair $(\mathcal{G}_1, \mathcal{G}_2)$ with node labels, the Maximum Common Subgraph (MCS) is the five-member ring structure highlighted in circle. The node-node correspondence in the MCS is drawn in dashed curves. Right: There are two nodes currently selected, and the rest of the two graphs is split into three bidomains. According to whether each node is connected to the two selected nodes or not, the nodes not in the current solution are split into three bidomains, denoted as "00", "01", and "10".

currently selected subgraph nodes $\mathcal{V}_{1s}$ and $\mathcal{V}_{2s}$, respectively. Figure 1 shows an example with three bidomains. Due to the subgraph isomorphism constraint posed by MCS, only nodes in $\mathcal{V}_1'$ can match to $\mathcal{V}_2'$ and vice versa. Thus, each bidomain can contribute at most $\min(|\mathcal{V}_1'|, |\mathcal{V}_2'|)$ nodes to the future best solution. Therefore, the upper bound can be estimated as $\sum_{\langle \mathcal{V}_1', \mathcal{V}_2' \rangle \in D} \min(|\mathcal{V}_1'|, |\mathcal{V}_2'|)$, where D denotes all the bidomains in the current state.

The search algorithm uses bidomains to not only compute upper bound for pruning unpromising branches, but also to select node pairs (denoted as "*policy*" in Algorithm 1). Since the bidomains naturally divide the unselected nodes into groups, when it comes to select which node pair to visit first, MCSPLIT and MCSPLIT+RL use additional heuristics to select the most promising bidomain and then select the most promising node pair in that bidomain. Details can be found in the appendix TODO.

3 PROPOSED METHOD

In this section we formulate the problem of MCS detection as learning an RL agent that iteratively grows the extracted subgraphs by adding new node pairs to the current subgraphs in a graph-structure-aware environment. We first describe the environment setup, then depict our proposed **J**oint **S**ubgraph-**N**ode (**JSNE**) network and the Deep Q-Network (DQN) which together provides actions for our agent to grow the subgraphs in a tree search context. We also describe how to leverage supervised data, when available, via imitation learning.

3.1 MCS DETECTION AS MARKOV DECISION PROCESS

Fundamentally different from image-based semantic graph matching Zanfir & Sminchisescu (2018) and other forms of graph alignment Xu et al. (2019a), graph matching for MCS detection yields two subgraphs that must be isomorphic to each other. Since subgraph isomorphism is a hard constraint the detection result must satisfy, instead of extracting two subgraphs in one shot, we design an RL agent which explores the input graph pair and sequentially grows the extracted two subgraphs one node pair at a time as shown in Figure 2. This not only allows the agent to capture the naturally occurring dependency between different extraction steps but also allows the environment to check whether two subgraphs are isomorphic across steps.

The iterative subgraph extraction process can be described by a Markov Decision Process $M = (\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S} = \{s_i\}$ is the set of states consisting of all the possible subgraphs extracted for input graph pairs, $\mathcal{A} = \{a_i\}$ is the set of actions representing the selection of new node pairs added to the current subgraphs, P is the transition dynamics that gives the transition probabilities between states, $p(s_{t+1}|s_t, ...s_0, a_t)$, which equals to $p(s_{t+1}|s_t, a_t)$ under the Markov property assumption, and $R(s_t)$ is the reward the agent receives after reaching s_t. The MCS extraction procedure can then be formulated as a sequence $(s_0, a_0, r_0, ..., s_n, a_n, r_n)$, where s_0 represents empty subgraphs and s_n represents the two final extracted subgraphs.

3.2 SUBGRAPH EXTRACTION ENVIRONMENT

In this section we give more details on the environment in which our RL agent extracts subgraphs from an input graph pair $(\mathcal{G}_1, \mathcal{G}_2)$.

3.2.1 STATE SPACE

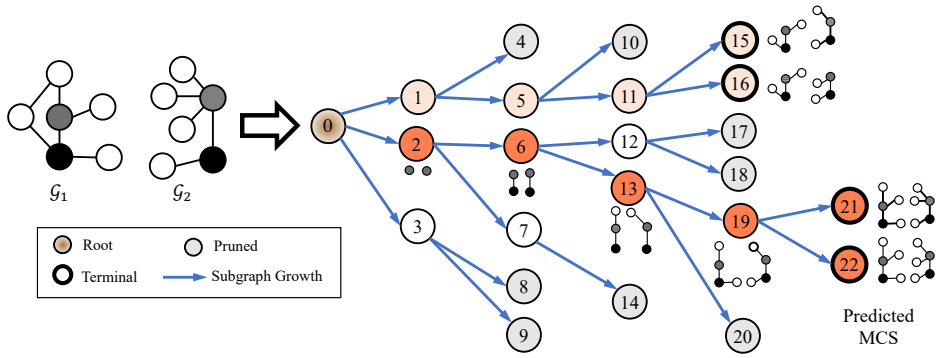

Figure 2: An overview of the proposed iterative MCS detection method. For $(\mathcal{G}_1, \mathcal{G}_2)$, RLMCS performs a beam-search-inspired subgraph extraction procedure (Section 3.4), yielding a tree structure where each tree node represents one state of the RL agent and each edge represents an action of growing the partial solution by one node for each of the two extracted subgraphs. During training, the JOINT SUBGRAPH-NODE EMBEDDING (Section 3.3.1) and DQN (Section 3.3.2) are trained under the reinforcement learning framework (Section 3.5) to predict actions at each state. The reward of transitioning from one state to another is +1 for growing each subgraph by one more node. During testing, the sequences leading to known MCS sizes are extracted e.g. (0,2,6,13,19,21) consisting of 5 steps and thus 5 tree nodes in each subgraph. The largest subgraph(s) will be returned as predicted MCS, corresponding to tree node 21 (and 22).

We define the state of the environment s_t as the intermediate extracted subgraphs $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ from input graph pair $(\mathcal{G}_1, \mathcal{G}_2)$ at time step t, which is fully observable by our RL agent. Figure 2 (c) shows an example graph pair from which the agent extracts subgraphs by following sequences which are part of the exploration tree which will be described in Section 3.4.

Proposition 3.1. For a given input graph pair $(\mathcal{G}_1, \mathcal{G}_2)$, the size of the state space is exponential in the input graph size, $|\mathcal{S}_{(\mathcal{G}_1,\mathcal{G}_2)}| \sim \mathcal{O}(2^{|\mathcal{V}_1|+|\mathcal{V}_2|})$.

Proof. At each step t, denote the size of the extracted two subgraphs as m_t, the whole graph sizes as $|\mathcal{V}_1| = n_1$ and $|\mathcal{V}_2| = n_2$. In this paper we consider connected induced subgraph, so the largest possible number of subgraphs extracted from one of the two input graphs of size n is $\binom{n}{m_t} = \frac{n!}{(n-m_t)!m_t!}$. For example, in two complete graphs with unlabeled nodes, choosing any m_t number of nodes along with the edges between these m_t nodes would lead to a valid connected induced subgraph. Similarly, the maximum amount of subgraph pairs $(\mathcal{G}_{1s}, \mathcal{G}_{2s})$ is $\binom{n_1}{m_t}\binom{n_2}{m_t}$.

According to Lemma 2.1, $|\text{MCS}(\mathcal{G}_1, \mathcal{G}_2)| \leq \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$. Denote $n_0 = \min(n_1, n_2)$. Since our RL agent grows one node pair at each step, i.e. $m_{t+1} = m_t + 1$, the initial subgraph is empty, i.e. $m_0 = 0$ and the final subgraph size at most is n_0, the total maximum amount of subgraph pairs that

could occur is

$$
\begin{aligned}
&\sum_{m_t=0}^{n_0} \binom{n_1}{m_t}\binom{n_2}{m_t} \\
&= \binom{n_1}{0}\binom{n_2}{0} + \binom{n_1}{1}\binom{n_2}{1} + ... + \binom{n_1}{n_0}\binom{n_2}{n_0} \\
&\leq \left(\binom{n_1}{0} + ... + \binom{n_1}{n_0}\right)\left(\binom{n_2}{0} + ... + \binom{n_2}{n_0}\right) \\
&\leq \left(\binom{n_1}{0} + ... + \binom{n_1}{n_1}\right)\left(\binom{n_2}{0} + ... + \binom{n_2}{n_2}\right) \\
&= 2^{n_1}2^{n_2} = 2^{n_1+n_2}
\end{aligned} \tag{1}
$$

□

3.2.2 Action Space

At any given step t, our RL agent maintains a subgraph for each of the two input graphs, denoted as $\mathcal{G}_{1s} = (\mathcal{V}_{1s}, \mathcal{E}_{1s})$ and $\mathcal{G}_{2s} = (\mathcal{V}_{2s}, \mathcal{E}_{2s})$. The agent "grows" both $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ by adding one new node pair as well as the induced edges between the new nodes and the selected nodes in the subgraphs, so $|\mathcal{V}_{1s}| = |\mathcal{V}_{2s}|$. Since MCS requires $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ to be connected as defined in Section 2, the action space can be reduced to only choosing the nodes that are directly connected to $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$.

Intuitively, these candidate nodes are at the "frontier" of the searching and subgraph-growing procedure, and are called "frontier" nodes. Formally, we define the candidate node sets in $\mathcal{G}_1$ and $\mathcal{G}_2$ to be $\mathcal{F}_1 = \{i \in \mathcal{V}_1 | i \notin \mathcal{V}_{1s}, \exists i' \in \mathcal{V}_{1s} : (i, i') \in \mathcal{E}_1\}$ and $\mathcal{F}_2 = \{j \in \mathcal{V}_2 | j \notin \mathcal{V}_{2s}, \exists j' \in \mathcal{V}_{2s} : (j, j') \in \mathcal{E}_2\}$. In Figure **??** (a), there are four frontier nodes highlighted in red color. For labeled graphs, since MCS requires node labels to match, we further reduce $\mathcal{F}_1 \times \mathcal{F}_2$ by removing the node pairs with different labels. We will discuss in detail the policy for selecting a node pair in Section 3.3.1.

3.2.3 Transition Dynamics

In the MCS detection environment, the MCS constraints impose rules that certain actions proposed by the agent must be rejected causing the state to remain unchanged. The subgraph connectivity constraint is ensured by restricting the candidate nodes to be from the frontier node sets as described in Section 3.2.2. However, the isomorphism constraint, i.e. the final extracted $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ must be isomorphic to each other, need to be checked by the environment, possibly via an exact or approximate graph isomorphism algorithm Cordella et al. (2001).

Leverage the fact that at each step t, $|\mathcal{G}_{1s}| = |\mathcal{G}_{2s}|$, and that graph isomorphism checking needs to be performed at each step, we propose our own **F**ast **I**terative **G**raph **I**somorphism (**FIGI**) algorithm which only incurs linear additional time in the number of nodes. At the root node, the two empty subgraphs are isomorphic. At step t, assuming $\mathcal{G}_{1s}$ is already isomorphic to $\mathcal{G}_{2s}$. Then, if we can simply check the new node pair proposed by the policy, and ensure $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ at $t+1$ are still isomorphic, the final result satisfies the isomorphism constraint by proof of induction. In implementation, this is achieved by maintaining a one-to-one node mapping at each step, $\mathcal{M}_t$, such that $\forall i \in \mathcal{V}_{1s}, \mathcal{M}_t(i) \in \mathcal{V}_{2s}$ and $\forall j \in \mathcal{V}_{2s}, \mathcal{M}_t^{-1}(j) \in \mathcal{V}_{1s}$. Specifically, we denote the proposed node pair as $(i', j') \in \mathcal{F}_1 \times \mathcal{F}_2$. We check if the nodes connected to i' and the nodes connected to j' match by first obtaining the nodes $\mathcal{V}_\alpha = \{i | i \in \mathcal{V}_{1s}, (i, i') \in \mathcal{E}_1\}$ and $\mathcal{V}_\beta = \{j | j \in \mathcal{V}_{2s}, (j, j') \in \mathcal{E}_2\}$ and check if $\forall i \in \mathcal{V}_\alpha, \mathcal{M}_t(i) \in \mathcal{V}_\beta$ and $\forall j \in \mathcal{V}_\beta, \mathcal{M}_t^{-1}(j) \in \mathcal{V}_\alpha$. Since the mapping $\mathcal{M}_t$ can be implemented as a hash table and iteratively updated, and the fact that the MCS size is bounded as in Lemma 2.1, in each iteration, the running time of **FIGI** is only in $\mathcal{O}(\min(|\mathcal{V}_1|, |\mathcal{V}_2|))$.

3.2.4 Reward

Since MCS detection aims to find the largest common subgraph, we define the reward our RL agent receives at each step t to be 1, i.e. $r(t) = 1$ for, $t = 1, 2, ..., T$, where T is the last step when the agent cannot further grow $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ without violating the MCS constraints, e.g. at the terminal tree

node 15, 16, 21, or 22 in Figure **??** (c). The RL agent is trained to achieve the largest accumulated reward, i.e. the predicted MCS size.

3.3 POLICY NETWORK

Having illustrated the graph generation environment, we outline the architecture of our policy network consisting of a graph embedding network and a Deep Q-Network (DQN) Mnih et al. (2015), which is learned by the RL agent to act in the environment. This DQN-based policy network takes the intermediate extracted subgraphs $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ as well as the original graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ as inputs, and outputs the action a_t, which predicts a new node pair, as described in Section 3.2.

3.3.1 EMBEDDING GENERATION: JSNE

Existing Graph Neural Networks (GNN) either aim to embed each node in a graph into a vector Duvenaud et al. (2015); Kipf & Welling (2016); Hamilton et al. (2017); Xu et al. (2019c) or an entire graph into a vector (typically via pooling) Ying et al. (2018); Zhang et al. (2018); Bai et al. (2019b); Hermsdorff & Gunderson (2019). However, for MCS detection, the node embeddings at each step t should be conditioned on the current extracted subgraphs $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$. What is worse, most GNNs embed for a single graph, with exceptions such as Graph Matching Networks (GMN) Zanfir & Sminchisescu (2018); Li et al. (2019), which however do not take subgraph information into account and run in at least quadratic time complexity $\mathcal{O}(|\mathcal{V}_1||\mathcal{V}_2|)$.

Here we present our JSNE network which jointly embed two graphs conditioned on the selected subgraphs in an elegant and efficient way within a unified model. As illustrated in Figure 2, at each step t, we add a "pseudo subgraph node" connected to every node in $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ to perform *graph comparison* via ***cross-graph communication*** through the psedudo node serving as an information-exchanging bridge. This resembles some earlier models Frasconi et al. (1998); Scarselli et al. (2009) which connects a "supersource" node to all the nodes in one graph to generate graph-level embedding. However, our goal is different and connects to $\mathcal{V}_{1s}$ and $\mathcal{V}_{2s}$ which grow across steps.

In each iteration, by adopting any existing message passing based GNN, e.g. GAT Velickovic et al. (2018), JSNE produces the embeddings of all the nodes and the subgraph jointly. Running in only $\mathcal{O}(|\mathcal{E}_1| + |\mathcal{E}_1|)$ time, JSNE approximates the quadratic GMN models which explicitly compare nodes in two graphs. Multiple JSNE layers are sequentially stacked similar to typical GNN architecture. Node embeddings are one-hot encoded initially according to their node labels (for unlabeled graphs a constant vector is used), and the subgraph embedding is initialized to zeros. The embeddings are iteratively updated as the subgraphs grow.

3.3.2 ACTION PREDICTION: DQN

Once the node and subgraph embeddings are generated by the multi-layered JSNE network, we use a Multilayer Perceptron (MLP) to produce a distribution of actions over the candidate node pairs $\mathcal{F}_1 \times \mathcal{F}_2$ (which is further reduced for labeled graphs as mentioned in Section 3.2.2), $\boldsymbol{q}$:

$$\boldsymbol{q} = \text{MLP}(\text{CONCAT}(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{h}_s, \boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2})), \tag{2}$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ denote node embeddings where $i \in \mathcal{F}_1$ and $j \in \mathcal{F}_2$, $\boldsymbol{h}_s$ denotes the subgraph embeddings , $\boldsymbol{h}_{\mathcal{G}_1}$ and $\boldsymbol{h}_{\mathcal{G}_2}$ denote graph-level embeddings generated by aggregating the node embeddings using an aggregation function such as $\text{SUM}(\cdot)$ and $\text{AVG}(\cdot)$.

3.4 SUBGRAPH EXPLORATION TREE

So far we have described how to generate a sequence of actions in RLMCS. However, according to Proposition 3.1, the search space is exponential in the graph sizes and is thus intractable to search thoroughly. Besides, for MCS detection, it is very likely to obtain suboptimal solution, i.e. predicted subgraph size smaller than the true MCS size, e.g. state node 15 and 16 in Figure 2, which is unwanted considering the nature of the task, though such suboptimal solutions still satisfy the the MCS constraints.

In order to address these issues, we propose a novel subgraph exploration tree, inspired by beam search, a dominant strategy for approximate decoding in structured prediction Negrinho et al. (2018) such as

machine translation Sutskever et al. (2014), and also a well-known graph search algorithm Baranchuk et al. (2019). With a hyperparameter budget BEAM SIZE, the agent is allowed to transition to at most BEAM SIZE number of best new states at any given state. In Figure 2, BEAM SIZE $= 3$, so each level of the tree can have up to 3 state nodes. The proposed algorithm is shown in Algorithm **??**.

Algorithm 1 Subgraph Exploration via Backtracking Tree Search

Input: Input graph pair $\mathcal{G}_1$, $\mathcal{G}_2$, budget.
Output: $maxSol$.
Initialize stack $\leftarrow$ `new` Stack().
Initialize $maxSol \leftarrow$ empty solution.
Initialize $i \leftarrow 0$.
stack.$push(s_0)$
while stack $\neq \emptyset$ **do**
  $i \leftarrow i + 1$;
  $s_t \leftarrow$ stack.$pop()$;
  $curSol \leftarrow s_t.getCurSol()$;
  if $|curSol| > |maxSol|$ **then**
    $maxSol \leftarrow curSol$;
  end if
  if $earlyStop(i, \text{budget})$ **then**
    break;
  end if
  $UB \leftarrow |curSol|$ + overestimate(s_t.bidomains);
  if $UB \leq |maxSol|$ **then**
    continue;
  end if
  $\mathcal{A}_t \leftarrow s_t$.actions;
  $a_t \leftarrow policy(s_t, \mathcal{A}_t)$;
  s_t.actions $\leftarrow s_t$.actions $\setminus \{a_t\}$;
  stack.$push(s_t)$
  $s_{t+1} \leftarrow$ env.update($s_t, \mathcal{A}_t$);
  stack.$push(s_{t+1})$
end while

Proposition 3.2. For a given input graph pair $(\mathcal{G}_1, \mathcal{G}_2)$, the maximum depth of the subgraph exploration tree is linear in the smaller of the two input graph sizes, $|\mathcal{T}_{(\mathcal{G}_1,\mathcal{G}_2)}| \sim \mathcal{O}(\min(|\mathcal{V}_1|, |\mathcal{V}_2|))$.

Proof. At each state node in the search tree, the intermediate subgraphs satisfy the MCS constraints by definition. Thus, the final predicted MCS size cannot be larger than the true MCS size $|\text{MCS}(\mathcal{G}_1, \mathcal{G}_2)|$. Defining the tree depth as the largest number of steps starting from root to terminal node, $|\mathcal{T}_{(\mathcal{G}_1,\mathcal{G}_2)}|$ is then equal to the predicted MCS size. Thus $|\mathcal{T}_{(\mathcal{G}_1,\mathcal{G}_2)}| \leq |\text{MCS}(\mathcal{G}_1, \mathcal{G}_2)| \leq \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$ according to Lemma 2.1. □

3.5 Overall Training

We adopt the standard Deep Q-learning framework Mnih et al. (2013). For each $(\mathcal{G}_1, \mathcal{G}_2)$, the agent performs the subgraph exploration tree search according to Algorithm **??**, after which the parameters in the JSNE and DQN are updated by performing mini-batch gradient descents over the mean squared error loss. Since imitation learning is known to help with training stability and performance Levine & Koltun (2013), we allow the agent to follow expert sequences generated by ground-truth MCS solvers, e.g. MCSPLIT McCreesh et al. (2017) during tree search, extending the exploration tree to an exploration-imitation tree with a hyperparameter ϵ denoting the percentage of pairs utilizing such labeled MCS instances. More details are shown in the Supplementary Material.

3.6 Complexity Analysis

At each state, the agent needs to generate embeddings and the q values which have worst-case time complexity $\mathcal{O}(|\mathcal{V}_1| * |\mathcal{V}_2|)$ due to the action space $|\mathcal{F}_1 \times \mathcal{F}_2|$ consisting of all the node pairs. Overall the tree depth is bounded by $\min(|\mathcal{V}_1|, |\mathcal{V}_2|)$ according to Proposition 3.2 and each level of the tree

has at most BEAM SIZE state nodes. Thus, the overall time complexity for each forward pass is $\mathcal{O}(\min(|\mathcal{V}_1|, |\mathcal{V}_2|) * \text{BEAM SIZE} * |\mathcal{V}_1| * |\mathcal{V}_2|)$. It is noteworthy that in contrast, state-of-the-art exact MCS computation algorithms Hoffmann et al. (2017); McCreesh et al. (2017) do not have worst-case time complexity guarantee, and as shown next, RLMCS strikes a good balance between speed and accuracy.

4 EXPERIMENTS

We evaluate RLMCS against two state-of-the-art exact MCS detection algorithms and a series of approximate graph matching methods from various domains. We conduct experiments on a variety of synthetic and real-world datasets. The code and datasets are provided as part of the Supplementary Material. All the baseline implementations are provided as well.

4.1 BASELINE METHODS

There are three groups of methods: Exact solvers including MCSPLIT McCreesh et al. (2017) and K↓ Hoffmann et al. (2017), supervised models including I-PCA Wang et al. (2019), GMN Li et al. (2019) and NEURALMCS Bai et al. (2020b), and unsupervised models including GW-QAP Xu et al. (2019a) and our proposed RLMCS.

MCSPLIT and K↓ are given a time budget of 100 seconds for each pair, whose results on training graph pairs are used to train the supervised model I-PCA, GMN and NEURALMCS. Specifically, the true MCS results are returned by the solvers as node-node mappings for nodes included in the MCS as illustrated in Figure **??**. For each graph pair, I-PCA, GMN and NEURALMCS are supervised models and generate a matching matrix $\boldsymbol{Y} \in \mathbb{R}^{|\mathcal{V}_1| \times |\mathcal{V}_2|}$ indicating the likelihood of each node pair being matched, which is fed into the binary cross entropy loss function against the true matching matrix generated by exact solvers, replacing their original domain-specific loss functions. GW-QAP performs Gromov-Wasserstein discrepancy Peyré et al. (2016) based optimization *for each graph pair* and outputs a matching matrix $\boldsymbol{Y}$, which does not require supervision by true MCS results.

During testing, necessary adaptation is performed to I-PCA, GMN and GW-QAP, which are designed for other tasks. Since they all yield a $\boldsymbol{Y}$ for each graph pair indicating the likelihood of node pairs being matching, we feed the predicted $\boldsymbol{Y}$ into our proposed subgraph exploration tree as detailed in Section 3.4. Specifically, we use Y_{ij} as the q value for node pair (i, j) instead of calling $GenActions()$ as in Algorithm **??**. All other aspects including the definition of frontier nodes, checking if a selection of node pair is allowed by environment, the value of BEAM SIZE, etc., are set the same way or value as our model RLMCS. A more detailed description can be found in the Supplementary Material.

Table 1: Comparison of MCS detection accuracy (%) on synthetic data generated by three models as well as real-world datasets. "Core" denotes the size of the "core graph", $\mathcal{G}_0$ as described in Section 4.4. Exact solvers are marked in superscript "*".

Method	**BA**		**ER**		**WS**		**REDDIT**	**AIDS**
	Core=32	Core=40	Core=32	Core=40	Core=32	Core=40		
MCSPLIT *	0*	0*	0*	0*	0*	0*	100*	100*
K↓ *	0*	0*	0*	0*	0*	0*	100*	100*
I-PCA	57.656	46.550	77.844	62.000	92.406	82.500	30.719	93.332
GMN	56.750	45.550	77.906	61.900	78.125	82.500	87.272	97.182
GW-QAP	35.938	27.825	16.688	13.075	40.625	40.425	34.618	57.919
NEURALMCS	57.188	47.256	77.094	61.000	84.375	80.000	**99.562**	**99.626**
RLMCS	**58.750**	**68.750**	**81.438**	**62.650**	**96.875**	**84.000**	93.187	98.467

4.2 PARAMETER SETTINGS

For our model, we utilize 3 layers of JSNE each with 64 dimensions for the embeddings. We use $\text{ReLU}(x) = \max(0, x)$ as our activation function. We set BEAM SIZE to 5. We ran all experiments with Intel i7-6800K CPU and one Nvidia Titan GPU. For DQN, we use MLP layers to project concatenated embeddings from 320 dimensions to a scalar. We observe better performance using

SUM$(\cdot)$ for real datasets and AVG$(\cdot)$ for synthetic datasets. For training, we set the learning rate to 0.001, the number of iterations to 600 for synthetic datasets and 2000 for real datasets, and use the Adam optimizer Kingma & Ba (2015). All experiments were implemented with the PyTorch and PyTorch Geometric libraries Fey & Lenssen (2019).

4.3 EVALUATION METRICS

For each testing graph pair, we collect the extracted subgraphs $\mathcal{G}_{1s}, \mathcal{G}_{2s}$, and measure the accuracy and running time (msec in wall time), which are then averaged across all the testing pairs. The definition of MCS detection accuracy is $\mathbb{1}(\mathcal{G}_{1s}, \mathcal{G}_{2s}) * \frac{m}{|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)|}$, where $m = |\mathcal{G}_{1s}| = |\mathcal{G}_{2s}|$. If $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ are returned within time limit and are connected, induced, as well as isomorphic to each, $\mathbb{1}(\mathcal{G}_{1s}, \mathcal{G}_{2s}) = 1$. Otherwise $\mathbb{1}(\mathcal{G}_{1s}, \mathcal{G}_{2s}) = 0$. When checking isomorphism, we set a timeout (10 seconds) for exact isomorphism checking Cordella et al. (2001), and switch to an approximate isomorphism checking[1] when timeout happens. We only use our proposed **FIGI** algorithm (Section 3.2.3) in the model, and do **not** use it in evaluation for $\mathbb{1}(\mathcal{G}_{1s}, \mathcal{G}_{2s})$ for fairness.

4.4 RESULTS ON SYNTHETIC DATA

The key property of RLMCS is its ability to extract MCS without much supervised data in an efficient and relatively accurate way. We generate three types of synthetic dataset using three types of popular graph generation models: Barabási-Albert (BA) Barabási & Albert (1999), Erdős-Rényi (ER) Gilbert (1959) and Watts–Strogatz (WS) Watts & Strogatz (1998). For each model, we first generate 1000 graph pairs for training and 100 for testing. When generating $(\mathcal{G}_1, \mathcal{G}_2)$, we first generate a "common core" $\mathcal{G}_0$ and then "grows" upon the core to obtain $\mathcal{G}_1$ and $\mathcal{G}_2$ where $|\mathcal{G}_1| = |\mathcal{G}_2| = 64$. We vary the core size to obtain several datasets. The exact procedure is shown in the Supplementary Material. This way, we obtain $(\mathcal{G}_1, \mathcal{G}_2)$ which we know $\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)$ must contain $\mathcal{G}_0$ but can be larger, essentially allowing the accuracy score to be above 1. Notice by incorporating the hard checking ($\mathbb{1}(\mathcal{G}_{1s}, \mathcal{G}_{2s})$ described in Section 4.3), all the predictions satisfy the MCS constraints, and the larger the accuracy, the larger the predicted subgraphs.

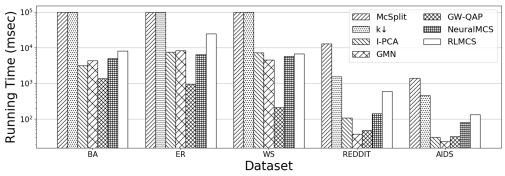

Figure 3: Running time comparison with y-axis in log scale. For synthetic datasets, we take the average running time across different core sizes.

We see that our model is both able to outperform baselines on instances where the ground truth MCS is difficult to obtain. The exact solvers fail to return within the time limit on all pairs, which is not surprising considering the NP-hard nature of the task and the lack of time complexity guarantee of MCSPLIT and K↓. Notice that in the original paper of MCSPLIT, the time limit is set to 1000 seconds (around 17 minutes per pair), an unaffordable time budget considering most machine learning models finish within 10 seconds as shown in Figure 3. The failure of exact solvers under the more realistic time budget setting implies that purely supervised models (I-PCA, GMN and NEURALMCS) cannot even be used in practice due to severe lack of labeled instances. In these experiments, we give these supervised models advantage by feeding $\mathcal{G}_0$ as "fuzzy" ground-truth MCS into their training stage. In practice, the most reasonable choice would be GW-QAP and our model RLMCS, with our model being much more accurate.

4.5 RESULTS ON REAL DATASETS

In addition to the synthetic datasets, we show that with the presence of supervision data, NEURALMCS is competitive in accuracy against baselines. We use two datasets: (1) REDDIT Yanardag &

[1] https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.isomorphism.could_be_isomorphic.html

Vishwanathan (2015) is a collection of 11929 online discussion threads represented as user interaction graphs, which is commonly used in graph classification tasks; (2) AIDS Zeng et al. (2009) is a popular dataset used in graph similarity computation and search. We observe MCSPLIT and K↓ fail to solve large graphs in these datasets most of the time, so we randomly random graphs less than 17 nodes forming 3556 graph pairs with average graph size being 11.8 nodes for REDDIT, and randomly sample 29610 graph pairs whose average graph size is 8.7 nodes for AIDS.

Notice that under this setting, RLMCS still uses **zero** labeled instances, i.e. purely unsupervised relying on the subgraph exploration and DQN training to extract MCS. In contrast, I-PCA, GMN, and NEURALMCS rely on the exact solvers to provide ground-truth instances. For these relatively small graphs, exact solvers successfully return the correct results under 100 seconds, but still much slower than machine learning approaches as shown in Figure 3. From Table 1, we see the heavy reliance on supervised data indeed brings performance gain to supervised models, especially NEURALMCS, and the unsupervised GW-QAP performs relatively poorly. However, our RL agent still yields the second best accuracy and performs better than I-PCA and GMN.

4.6 CONTRIBUTION OF EACH MODULE

4.6.1 CONTRIBUTION OF JSNE

We replace the JSNE module with two other graph embedding modules to study the effect of JSNE. Specifically, we use GAT Velickovic et al. (2018) and GMN Li et al. (2019). As shown in Table 2, with GAT and GMN, our agent can no longer perform conditional embeddings based on subgraph extraction status, and instead uses the same embeddings without considering subgraph growth, leading to worse performance.

Table 2: Contribution of JSNE to the accuracy of RLMCS on REDDIT. "w/" denotes "with"; "w/" denotes "without".

Method	**Accuracy**
RLMCS w/ GAT	82.685
RLMCS w/ GMN	72.026
RLMCS w/ JSNE (full)	**93.187**

4.6.2 CONTRIBUTION OF SUBGRAPH EXPLORATION WITH SEARCH TREE

As mentioned in Section 4.1, we adapt the baseline models to tackle MCS detection via feeding the matching matrix $\boldsymbol{Y}$ matrix into our Algorithm **??**. To investigate the effectiveness of our search tree, we feed $\boldsymbol{Y}$ generated by these methods into a simpler strategy based on thresholding and Hungarian Algorithm Kuhn (1955): We remove nodes with overall matching scores computed as summation across rows and columns of $\boldsymbol{Y}$ lower than a tunable threshold. Since the number of remaining nodes in $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ may be different, we then run the Hungarian Algorithm for Linear Assignment Problems (LAP) on $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$, yielding two subgraphs of the same size returned as the prediction. We also try running the Hungarian Algorithm on the original $\boldsymbol{Y}$ to obtain two subgraphs as final prediction, and report the better of the two results.

It is noteworthy that NEURALMCS uses its own search method called Guided Subgraph Extraction (GSE) Bai et al. (2020b), which can be roughly considered as a simpler variant of our proposed search with BEAM SIZE $= 1$. More details on comparing these strategies can be found in the Supplementary Material. As shown in Table 3, with this simpler alternative strategy, the performance of GRAPH MATCHING NETWORKS and NEURALMCS drops by a large amount, while the performance of I-PCA and GW-QAP increases slightly.

4.6.3 CONTRIBUTION OF FIGI

The Fast Iterative Graph Isomorphism (FIGI) algorithm proposed in Section 3.2.3 is used by the environment to check if a selection of new node pair is allowed. Here we compare FIGI with two alternatives: (1) The exact graph isomorphism used in evaluation as described in Section 4.3; (2) The Subgraph Isomorphism Network (SIN) proposed in NEURALMCS Bai et al. (2020b) which essentially performs Weisfeiler-Lehman (WL) graph isomorphism test Shervashidze et al. (2011) using node embeddings generated at each step t. As shown in Table 4, on REDDIT FIGI successfully ensures all the returned predictions satisfy the isomorphism constraint posed by the MCS definition, and is

Table 3: Contribution of Subgraph Exploration Tree to the accuracy of RLMCS on REDDIT.

Method	Accuracy
I-PCA w/o Search	33.987
I-PCA w/ Search	30.719
GMN w/o Search	43.137
GMN w/ Search	87.272
GW-QAP w/o Search	35.948
GW-QAP w/ Search	34.618
NEURALMCS w/o Search	29.669
NEURALMCS w/ GSE	99.562

much faster than other approaches. In fact, we observe that under all the settings in our experiments, FIGI exhibits perfect isomorphism detection accuracy. A more detailed discussion on FIGI in the Supplementary Material.

Table 4: Contribution of FIGI to the isomorphism percentage (Iso %) running time (msec) of RLMCS on REDDIT.

Method	Iso %	Running Time
RLMCS w Exact GI	100	1014.945
RLMCS w/ SIN GI	100	665.457
RLMCS w/ FIGI	100	594.687

5 RELATED WORK

MCS detection is HP-hard, with existing methods based on constraint programming (Vismara & Valery, 2008; McCreesh et al., 2016), branch and bound (McCreesh et al., 2017; Liu et al., 2019), mathematical programming (Bahiense et al., 2012), conversion to maximum clique detection (Levi, 1973; McCreesh et al., 2016), etc. Closed related to MCS detection is Graph Edit Distance (GED) computation (Bunke, 1983), which in the most general form refers to finding a series of edit operations that transform one graph to another and has also been adopted in many task where the matching or similarity between graphs is necessary. There is a growing trend of using machine learning approaches to approximate graph matching and similarity score computation, but these works either do not address MCS detection specifically and must be adapted Zanfir & Sminchisescu (2018); Wang et al. (2019); Yu et al. (2020); Xu et al. (2019b;a); Bai et al. (2019a; 2020a); Li et al. (2019); Ling et al. (2020), or rely on labeled instances Bai et al. (2020b)

6 CONCLUSION AND FUTURE WORK

We have proposed a reinforcement learning method which unifies graph representation learning, deep Q-learning and imitation learning into a single framework. We show that the resulting model shows superior performance on various graph datasets. In the future, we plan to extend our method to subgraph matching Sun et al. (2012), which requires the matching and retrieval of all subgraphs contained in a large graph. Additionally, to improve the scalability of our method, we will explore new graph search and matching algorithms.

REFERENCES

Laura Bahiense, Gordana Manić, Breno Piva, and Cid C De Souza. The maximum common edge subgraph problem: A polyhedral investigation. *Discrete Applied Mathematics*, 160(18):2523–2541, 2012.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. *WSDM*, 2019a.

Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. Unsupervised inductive whole-graph embedding by preserving graph proximity. *IJCAI*, 2019b.

Yunsheng Bai, Hao Ding, Ken Gu, , Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. *AAAI*, 2020a.

Yunsheng Bai, Derek Xu, Ken Gu, Xueqing Wu, Agustin Marinovic, Christopher Ro, Yizhou Sun, and Wei Wang. Neural maximum common subgraph detection with guided subgraph extraction, 2020b. URL `https://openreview.net/forum?id=BJgcwh4FwS`.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. Learning to route in similarity graphs. *ICML*, 2019.

Horst Bunke. What is the distance between graphs. *Bulletin of the EATCS*, 20:35–39, 1983.

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3-4):255–259, 1998.

Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *2011 31st International Conference on Distributed Computing Systems*, pp. 393–402. IEEE, 2011.

Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pp. 149–159, 2001.

Edmund Duesbury, John Holliday, and Peter Willett. Comparison of maximum common subgraph isomorphism algorithms for the alignment of 2d chemical structures. *ChemMedChem*, 13(6): 588–598, 2018.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pp. 2224–2232, 2015.

Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.

Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pp. 1024–1034, 2017.

Gecia Bravo Hermsdorff and Lee Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. In *NeurIPS*, pp. 7734–7745, 2019.

Ruth Hoffmann, Ciaran McCreesh, and Craig Reilly. Between subgraph isomorphism and maximum common subgraph. In *AAAI*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2016.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Giorgio Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341, 1973.

Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, pp. 1–9, 2013.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *ICML*, 2019.

Yongjiang Liang and Peixiang Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*, pp. 783–794. IEEE, 2017.

Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Chunming Wu, and Shouling Ji. Hierarchical graph matching networks for deep graph similarity learning, 2020. URL `https://openreview.net/forum?id=rkeqn1rtDH`.

Yan-li Liu, Chu-min Li, Hua Jiang, and Kun He. A learning based branch and bound for maximum common subgraph problems. *IJCAI*, 2019.

Ciaran McCreesh, Samba Ndojh Ndiaye, Patrick Prosser, and Christine Solnon. Clique and constraint models for maximum common (connected) subgraph problems. In *International Conference on Principles and Practice of Constraint Programming*, pp. 350–368. Springer, 2016.

Ciaran McCreesh, Patrick Prosser, and James Trimble. A partitioning algorithm for maximum common subgraph problems. 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *NeurIPS Deep Learning Workshop 2013*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Renato Negrinho, Matthew Gormley, and Geoffrey J Gordon. Learning beam search policies via imitation learning. In *NeurIPS*, pp. 10652–10661, 2018.

Younghee Park, Douglas S Reeves, and Mark Stamp. Deriving common malware behavior through graph clustering. *Computers & Security*, 39:419–430, 2013.

Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *ICML*, pp. 2664–2672, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12(Sep):2539–2561, 2011.

Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. Efficient subgraph matching on billion node graphs. *VLDB*, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NeurIPS*, pp. 3104–3112, 2014.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

Philippe Vismara and Benoît Valery. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In *International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences*, pp. 358–368. Springer, 2008.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. *ICCV*, 2019.

Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *ICDE*, pp. 210–221. IEEE, 2012.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440, 1998.

Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. In *NeurIPS*, pp. 3046–3056, 2019a.

Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin. Gromov-wasserstein learning for graph matching and node embedding. *ICML*, 2019b.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019c.

Xifeng Yan, Philip S Yu, and Jiawei Han. Substructure similarity search in graph databases. In *SIGMOD*, pp. 766–777. ACM, 2005.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, pp. 1365–1374. ACM, 2015.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *NeurIPS, pages=6410–6421, year=2018*.

Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In *ICLR*, 2020. URL `https://openreview.net/forum?id=rJgBd2NYPH`.

Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *CVPR*, pp. 2684–2693, 2018.

Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS, pages=5165–5175, year=2018*.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

Xinyi Zhang and Lihui Chen. Capsule graph neural network. *ICLR*, 2019.

Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. A partition-based approach to structure similarity search. *PVLDB*, 7(3):169–180, 2013.

Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Graph similarity search with edit distance constraint in large graph databases. In *CIKM*, pp. 1595–1600. ACM, 2013.

A Dataset Description

We run experiments on 4 diverse different real world datasets coming from the chemical, programming language, and social network domains. For each dataset, we split the pairs into training, validation, and testing sets in the ratio of 6:2:2 such that none of the training, validation, or testing sets share any common graphs. For each dataset, we either one-hot encode node labels, if the dataset has node labels, or provide the same initial encoding, if the dataset does not have node labels. The code and the datasets have been published to this anonymous link: `https://github.com/openpublicforpapers/NeuralMCS`.

A.1 AIDS

AIDS is a dataset of antivirus screen chemical compounds, coming from the Developmental Therapeutics Program at NCI/NIH[2]. The AIDS dataset has been used by various works in graph matching (Zeng et al., 2009; Wang et al., 2012; Zheng et al., 2013; Zhao et al., 2013; Liang & Zhao, 2017; Bai et al., 2019a). These graphs consist of labeled nodes and unlabeled edges, where nodes represent chemical elements (ex. Carbon, Nitrogen, Chlorine, and etc.) and edges represent bonds between atoms. There are a total of 700 graphs, from which we sample 29610 graph pairs. The average graph size is 8.664 with the largest graph having 10 nodes.

A.2 LINUX

LINUX is a dataset of program dependence graphs (PDG) describing individual functions generated from the Linux kernel (Wang et al., 2012). These graphs consist of unlabeled nodes and unlabeled edges, where nodes represent statements and edges represent control flow between statements. There are a total of 1000 graphs, from which we sample 60114 graph pairs. The average graph size is 7.591 and the largest graphs have 10 nodes.

A.3 IMDB

IMDB is a dataset of ego-networks of movie actors/actresses from the IMDB website (Yanardag & Vishwanathan, 2015). This dataset has been used by various works in graph classification (Zhang & Chen, 2019; Bai et al., 2019b). These graphs consist of unlabeled nodes and unlabeled edges, where nodes represent actors/actresses and edges represent whether they have had collaborations. There are a total of 1500 graphs, from which we sample 135702 graph pairs. The average graph size is 12.981 and the largest graphs have 89 nodes.

A.4 REDDIT

REDDIT is a dataset of online dicussion networks from the Reddit online discussion website (Yanardag & Vishwanathan, 2015). These graphs consist of unlabeled nodes and unlabeled edges, where nodes represent users and edges represent whether users have responded to eachother's comments. There are a total of 7112 graphs, from which we sample 3556 pairs. The average graph size is 11.8 nodes and the largest graph has 16 nodes.

[2] https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data