LEARNING TO SEARCH FOR FAST 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 MCS solvers rely on heuristics in search which in practice cannot find good solution for large graph pairs under a limited search budget. Here we propose GLSEARCH, a Graph Neural Network based model for MCS detection, which learns to search. Our model uses a state-of-the-art branch and bound algorithm as the backbone search algorithm to extract subgraphs by selecting one node pair at a time. In order to make better node selection decision at each step, we replace the node selection heuristics with a novel task-specific Deep Q-Network (DQN), allowing the search process to find larger common subgraphs faster. To enhance the training of DQN, we leverage the search process to provide supervision in a pre-training stage and guide our agent during an imitation learning stage. Therefore, our framework allows search and reinforcement learning to mutually benefit each other. Experiments on synthetic and real-world large graph pairs demonstrate that our model outperforms state-of-the-art MCS solvers and neural graph matching network models.

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 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 1), is an important yet particularly hard task. MCS naturally encodes the degree of similarity between two graphs, is domain-agnostic, and thus has occurred 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).

MCS detection is NP-hard in its nature and is thus a very challenging task. On one hand, the state-of-the-art exact MCS detection algorithms based on branch and bound run in exponential time in worst cases (Liu et al., 2019). What is worse, they rely on several heuristics on how to explore the search space. For example, MCSP (McCreesh et al., 2017) uses node degree as its heuristic by choosing high-degree nodes to visit first, but in many cases the true MCS contains small-degree nodes. On the other hand, existing machine learning approaches to graph matching such as Wang et al. (2019) and Bai et al. (2020b) 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.

In this paper, we present GLSEARCH (Graph Learning to Search), a general framework for MCS detection combining the advantages of search and reinforcement learning. GLSEARCH learns to search by adopting a Deep Q-Network (DQN) (Mnih et al., 2015) to replace the node selection heuristics required by state-of-the-art MCS solvers, leading to faster arrival of the optimal solution for an input graph pair, which is particularly useful when the simpler heuristics fail and graphs are large

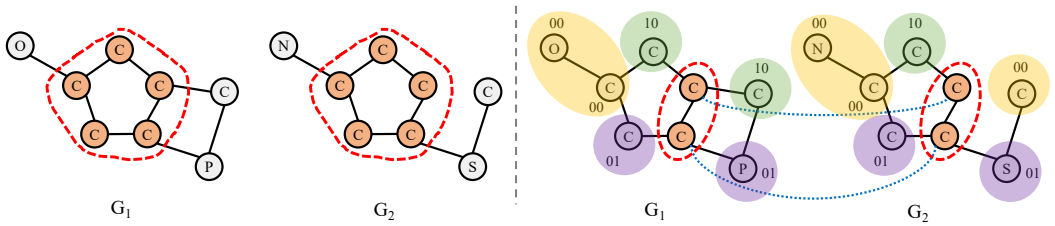


Figure 1: Left: For graph pair $(\mathcal{G}_1, \mathcal{G}_2)$ with node labels, the induced connected MCS is the five-member ring structure highlighted in circle. Right: At this step, there are two nodes currently selected. According to whether each node is connected to the two selected nodes or not, the nodes not in the current solution are split into three bidomains (Section 2.2), denoted as "00", "01", and "10", where "0" indicates not connected to a node in the selected two nodes, and "1" indicates connected.

with a limited search budget. Thanks to the learning capacity of Graph Neural Networks (GNN), our DQN is specially designed for the MCS detection task with a novel reformulation of DQN to better capture the effect of different node selections. Given the large action space incurred by large graph pairs, to enhance the training of DQN, we leverage the search algorithm to not only provide supervised signals in a pre-training stage but also offer guidance during an imitation learning stage.

Experiments on real graph datasets that are significantly larger than exisitng datasets adopted by state-of-the-art MCS solvers demonstrate that GLSEARCH outperforms baseline solvers and machine learning models for graph matching in terms of effectiveness by a large margin. Our contributions can be summarized as follows:

- We address the challenging yet important task of Maximum Common Subgraph detection for general-domain input graph pairs and propose GLSEARCH as the solution.
- The key novelty is the DQN which learns to search. Specifically, it is trained under the reinforcement learning framework to make the best decision at each search step in order to quickly find the best MCS solution during search. The search in turns helps training of DQN in a pre-training stage and an imitation learning stage.
- We conduct extensive experiments on medium-size synthetic graphs and very large real-world graphs to demonstrate the effectiveness of the proposed approach compared against a series of string baselines in MCS detection and graph matching.

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 in Figure 1.

Graph isomorphism and subgraph isomorphism can be regarded as two special tasks of MCS: $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| = |\mathcal{V}_1| = |\mathcal{V}_2|$ if $\mathcal{G}_1$ are isomorphic to $\mathcal{G}_2$, $|\mathrm{MCS}(\mathcal{G}_1, \mathcal{G}_2)| = \min(|\mathcal{V}_1|, |\mathcal{V}_2|)$ when $\mathcal{G}_1$ is subgraph isomorphic to $\mathcal{G}_2$ or $\mathcal{G}_2$ is subgraph isomorphic to $\mathcal{G}_1$.

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, MCSP, is presented in McCreesh et al. (2017) and the more advanced

version, MCSP+RL, is proposed in Liu et al. (2019). The whole search algorithm, outlined in Algorithm 1[1], is a branch-and-bound algorithm that maintains a best solution found so far throughout the search, which is initialized as empty subgraphs. In each search iteration, denote the current search state as s_t consisting of $\mathcal{G}_1, \mathcal{G}_2$, the current selected subgraphs $\mathcal{G}_{1s} = (\mathcal{V}_{1s}, \mathcal{E}_{1s})$ and $\mathcal{G}_{2s} = (\mathcal{V}_{2s}, \mathcal{E}_{2s})$ as well as their node-node mappings. The algorithm tries to select one node pair, (v_i, v_j), where v_i is from $\mathcal{G}_1$ and v_j is from $\mathcal{G}_2$, as its action, denoted as a_t, and either backtracks to the parent search state if the solution is not promising or continues the search otherwise. Various heuristics on node pair selection policy, denoted as "$policy$", are proposed in MCSP and MCSP+RL. For example, in MCSP, nodes of large degrees are selected before small-degree nodes.

There are two major limitations of MCSP and MCSP+RL: (1) Such heuristics-based node pair selection policy cannot adapt to different graph structures[2]; (2) The search may enter a bad state and get "stuck" without finding a better (larger) solution, $maxSol$, for many iterations.

Algorithm 1 Branch and Bound for MCS

Input: Input graph pair $\mathcal{G}_1, \mathcal{G}_2$.
Output: $maxSol$.
Initialize stack $\leftarrow$ `new` Stack().
Initialize $maxSol \leftarrow$ empty solution.
stack.$push(s_0)$;
while stack $\neq \emptyset$ **do**
  $s_t \leftarrow$ stack.$pop()$;
  $curSol \leftarrow s_t.getCurSol()$;
  if $|curSol| > |maxSol|$ **then**
    $maxSol \leftarrow curSol$;
  end if
  $UB_t \leftarrow |curSol|$ + overestimate(s_t);
  if $UB_t \leq |maxSol|$ **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

At each search state, in order to compute the upper bound "UB_t" and reduce the action space "$\mathcal{A}_t$", i.e. the candidate node pairs to select from, the concept of "bidomain" is introduced. Bidomains partition the nodes in the remaining subgraphs, i.e. outside $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$, into equivalent classes. A bidomain D_k consists of two sets of nodes, $\langle \mathcal{V}'_{k1}, \mathcal{V}'_{k2} \rangle$ where $\mathcal{V}'_{k1}$ and $\mathcal{V}'_{k2}$ have the same connectivity pattern with respect to the already matched nodes $\mathcal{V}_{1s}$ and $\mathcal{V}_{2s}$. Figure 1 shows an example with three bidomains. Due to the subgraph isomorphism constraint posed by MCS, only nodes in $\mathcal{V}'_{k1}$ can match to $\mathcal{V}'_{k2}$ and vice versa. This also guarantees the extracted subgraphs at each state are isomorphic to each other. Thus, each bidomain can contribute at most $\min(|\mathcal{V}'_{k1}|, |\mathcal{V}'_{k2}|)$ nodes to the future best solution. Therefore, the upper bound can be estimated as $\sum_{D_k \in \mathcal{D}} \min(|\mathcal{V}'_{k1}|, |\mathcal{V}'_{k2}|)$, where $\mathcal{D}$ denotes all the bidomains in the current state. This upper bound computation is consistently used for all the methods in the paper. The major difference is in the policy for node pair selection.

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 Deep Q-Network (DQN) which provides actions for our agent to grow the subgraphs in a search context. We also describe how to leverage supervised data via pre-training and imitation learning.

3.1 Leveraging DQN for Search

Since graph matching for MCS detection must satisfy a hard constraint that the resulting two subgraphs must be isomorphic to each other, instead of learning to match two graphs in one shot, we design an RL agent which explores the input graph pair and sequentially grows the extracted

[1]The original algorithm is recursive. To highlight our novelty, we rewrite into an equivalent iterative version.

[2]MCSP+RL claims it uses reinforcement learning (RL) to compute a score for each node pair. However, the scores for each node are the only learnable parameters, whose update relies on a heuristic. This requires the learning of scores for each new testing graph pair. In contrast, we use continuous embeddings to represent graphs which are fed into a DQN to compute a score trained on many graph pairs. Once trained, GLSEARCH can be applied to any new testing pair without retraining.

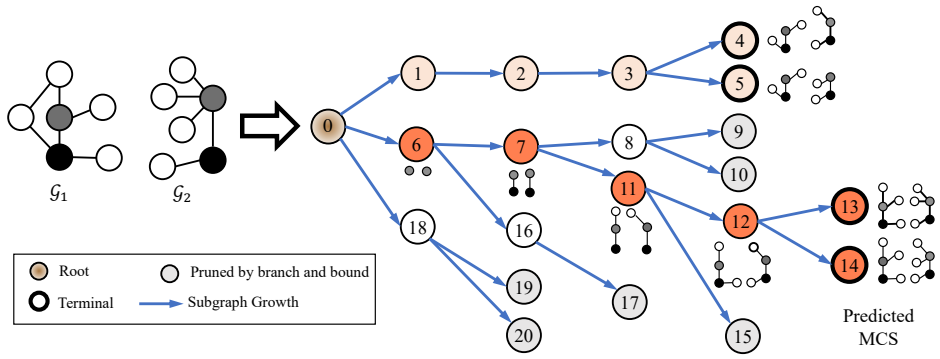


Figure 2: An illustration of the search process for MCS detection. For $(\mathcal{G}_1, \mathcal{G}_2)$, the branch and bound search algorithm (Section 2.2 and Algorithm 1) yields a tree structure where each node represents one state (s_t) with id reflecting the order in which states are visited, and each edge represents an action (a_t) of selecting one more node pair. The search is essentially depth-first with pruning by the upper bound check. Our model learns the node pair selection strategy, i.e. which state to visit first. If state 6 can be visited before state 1, a large solution can be found in less iterations. When the search completes or a pre-defined search iteration budget is used up, the best solution will be returned, corresponding to state 13 (and 14).

two subgraphs one node pair at a time. The iterative subgraph extraction process can be described by a Markov Decision Process (MDP), where the definitions of state and action are the same as Section 2.2. The difference is that, for MDP, reward needs to be defined too. for MCS, the immediate reward for transitioning from one state to any next state is $+1$ since one new node pair is selected.

To address the issue that the algorithm may get stuck in a bad state for many iterations without finding a larger solution, we utilize additional information stored in Q-values computed by our learned model. We suppose backtracking to an earlier better state can alleviate such issue in practice, but there lacks a principled measure for MCSP and MCSP+RL to determine which earlier state is better. By design, our node pair selection policy is a learned DQN, so our agent knows not only the quality of immediate actions, but also the values associated with previous states. Therefore, if the best solution found so far does not increase, i.e. we do not enter line 10 of Algorithm 1 for a pre-defined number of iterations, in the next iteration, we find the best state as determined by the Bellman Equation, remove that state, then visit it on line 7. We refer to this improved search methodology as **promise-based search**. More details can be found in the supplementary material.

3.2 REPRESENTATION LEARNING FOR DQN

Since the action space can be large for MCS, we leverage the representation learning capacity of continuous representations for DQN design. At state s_t, for each action a_t, our DQN predicts a $Q(s_t, a_t)$ representing the future reward to go if the action $a_t = (i, j)$ where $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$ is selected, intuitively corresponding to the largest number of nodes that will be eventually selected starting from the action edge (s_t, a_t) as in tree in Figure 2.

Based on the above insights, one can design a simple DQN leveraging the representation learning power of Graph Neural Networks (GNN) such as Kipf & Welling (2016) and Velickovic et al. (2018) by passing $\mathcal{G}_1$ and $\mathcal{G}_2$ to a GNN to obtain one embedding per node, $\{\boldsymbol{h}_i | \forall i \in \mathcal{V}_1\}$ and $\{\boldsymbol{h}_j | \forall j \in \mathcal{V}_2\}$. Denote CONCAT as concatenation, READOUT as a readout operation that aggregates node-level embeddings into subgraph embeddings $\boldsymbol{h}_{s1}$ and $\boldsymbol{h}_{s1}$, and whole-graph embeddings $\boldsymbol{h}_{\mathcal{G}_1}$ and $\boldsymbol{h}_{\mathcal{G}_2}$. A state can then be represented as $\boldsymbol{h}_{s_t} = \text{CONCAT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2}, \boldsymbol{h}_{s1}, \boldsymbol{h}_{s2})$. An action can be represented as $\boldsymbol{h}_{a_t} = \text{CONCAT}(\boldsymbol{h}_i, \boldsymbol{h}_j)$. The Q function can the be designed as:

$$Q(s_t, a_t) = \text{MLP}\big(\text{CONCAT}(\boldsymbol{h}_{s_t}, \boldsymbol{h}_{a_t})\big) = \text{MLP}\big(\text{CONCAT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2}, \boldsymbol{h}_{s1}, \boldsymbol{h}_{s2}, \boldsymbol{h}_i, \boldsymbol{h}_j)\big). \quad (1)$$

However, there are several flaws to this simple design of Q function:

(A) $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ generated by typical GNNs encode only local neighborhood information, but $Q(s_t, a_t)$ represents the long-term effect of adding (i, j). What is worse, different node pairs have different embeddings, but their immediate rewards are always $+1$ in MCS.

(B) Swapping the order of $\mathcal{G}_1$ and $\mathcal{G}_2$ should not cause $Q(s_t, a_t)$ to change, but concatenating embeddings from the two graphs causes the DQN to be sensitive to their ordering.

(C) How to effectively leverage the node-node mappings between $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ for predicting $Q(s_t, a_t)$ remains a challenge.

To address these issues, we propose the following improvements over the simple DQN design.

Factoring out Action In order to maximally reflect the effect of adding node pair (i, j) to $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$, we first notice that $Q^*(s_t, a_t) = r_t + \gamma V^*(s_{t+1}) = 1 + \gamma V^*(s_{t+1})$ in MCS, where Q and V are the Q and value functions, respectively, and γ is the discount factor. Then, in order to compute the effect of a_t, we can compute the value associated with s_{t+1} which does not depend on a_t and avoids the use of local $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$.

Interaction between Input Graphs To resolve the graph symmetry issue, we first construct the interaction between the embeddings from two graphs, i.e. $\text{INTERACT}(\boldsymbol{h}_{x1}, \boldsymbol{h}_{x2})$, where $\boldsymbol{h}_{x1}$ and $\boldsymbol{h}_{x2}$ represent any embedding from $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, and then concatenate the interacted embeddings which are fed into the final MLP. In implementation, various interaction operators such as addition, element-wise multiplication, max pooling, etc. can be adopted.

Bidomain Representations Bidomains are derived from node-node mappings and partition the rest of $\mathcal{G}_1$ and $\mathcal{G}_2$, which is a more useful signal for predicting the future reward. In fact, as described in Section 2.2, bidomains have been adopted to in search-based MCS solvers to estimate the upper bound. Here, we require the harder prediction of $Q(s_t, a_t)$ for which we propose to use the representation of bidomains. Denote $\boldsymbol{h}_{D_k}$ as the representation for bidomain D_k. Similar to computing the graph-level and subgraph-level embeddings, we compute $\boldsymbol{D}_k$ as

$$\boldsymbol{h}_{D_k} = \text{INTERACT}\big(\text{READOUT}(\{\boldsymbol{h}_i | i \in \mathcal{V}'_{k1}\}), \text{READOUT}(\{\boldsymbol{h}_j | j \in \mathcal{V}'_{k2}\})\big). \tag{2}$$

Since we require the MCS to be connected subgraphs, we differentiate bidomains $\mathcal{D}^{(c)}$ that are connected (adjacent) to $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ from the single bidomain D_0 disconnected (unconnected) from $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ (e.g. bidomain "00" in Figure 1). Given all the bidomain embeddings, we compute a single representation for $\mathcal{D}^{(c)}$, $\boldsymbol{h}_{\mathcal{D}c} = \text{READOUT}(\{\boldsymbol{h}_{D_k} | k \in \mathcal{D}^{(c)}\})$. Our final DQN has the form:

$$Q(s_t, a_t) = 1 + \gamma \text{MLP}\Big(\text{CONCAT}\big(\text{INTERACT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2}), \text{INTERACT}(\boldsymbol{h}_{s1}, \boldsymbol{h}_{s2}), \boldsymbol{h}_{\mathcal{D}c}, \boldsymbol{h}_{\mathcal{D}0}\big)\Big). \tag{3}$$

3.3 Leveraging Search for DQN Training

At each state s_t, the action space size in the worst case is quadratic to the number of nodes in the remaining subgraphs. Thus, to enhance the training of our DQN, before the standard training of DQN (Mnih et al., 2013), we pre-train DQN and guide its exploration with expert trajectories supplied by the search algorithm.

For the pre-training stage, we first observe the overall mse loss is $(y_t - Q(s_t, a_t))^2$ where y_t the target for iteration t and $Q(s_t, a_t)$ is the predicted $Q(s_t, a_t)$. We then notice that for small training graph pairs, the complete exploration of search space can be performed to obtain the true target for every (s_t, a_t) by finding the longest sequence starting from s_t to a leaf node in the search tree.

For larger graph pairs though, finding the true target becomes too slow. In that case, after pre-training, we enter the imitation learning stage where we let the agent mimic the decision made by the state-of-the-art MCS search algorithm instead of relying on its own predicted $Q(s_t, a_t)$. More details can be in found in the supplementary material.

4 Experiments

We evaluate GLSearch 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 medium-sized synthetic graph datasets and large real-world graph datasets, whose details can be found in the supplementary material. Among the different baseline models, we find no consistent trend. This indicates the difficulty of our task, as existing methods can not find a consistent policy that guarantees good performance on datasets from different domains. Our model can substantially outperform the baselines, highlighting the significance of our contributions to learning for search.

Table 1: Results on synthetic graphs. Each dataset consists of 50 randomly generated pairs labeled as "⟨generation algorithm⟩-⟨number of nodes in each graph⟩". "BA", "ER", and "WS" refer to the Barabási-Albert (BA) (Barabási & Albert, 1999), the Erdős-Rényi (ER) (Gilbert, 1959), and the Watts–Strogatz (WS) (Watts & Strogatz, 1998) algorithms, respectively. We show the ratio of the (average) size of the subgraphs found by each method with respect to the best result on that dataset.

Method	**BA-50**	**BA-100**	**ER-50**	**ER-100**	**WS-50**	**WS-100**
MCSP	0.913	0.892	0.842	0.896	0.905	0.856
MCSP+RL	0.923	0.857	0.844	0.877	0.913	0.875
GW-QAP	0.945	0.887	0.855	0.925	0.916	0.898
I-PCA	0.899	0.863	0.848	0.923	0.879	0.852
NEURALMCS	0.908	0.889	0.846	0.906	0.889	0.865
GLSEARCH-RAND	0.995	0.987	0.920	0.978	0.967	0.931
GLSEARCH	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
BEST SOLUTION SIZE	19.12	34.38	26.56	37.64	29.48	55.56

4.1 BASELINE METHODS

There are two groups of methods: Exact MCS algorithms including MCSP (McCreesh et al., 2017) and MCSP+RL (Liu et al., 2019), learning based graph matching models including GW-QAP (Xu et al., 2019a), I-PCA (Wang et al., 2019), and NEURALMCS (Bai et al., 2020b).

All the methods either originally use or are adapted to use the branch and bound search framework in Section 2.2 with differences in node pair selection policy and training strategies. GW-QAP performs Gromov-Wasserstein discrepancy (Peyré et al., 2016) based optimization *for each graph pair* and outputs a matching matrix $\boldsymbol{Y}$ for all node pairs indicating the likelihood of matching which is treated the same way as our q scores, i.e. at each search iteration we index into $\boldsymbol{Y}$ to select a node pair. I-PCA and NEURALMCS also output a matching matrix but require supervised training, and thus are trained using the same training data graph pairs as our GLSEARCH but with different loss functions and training signals. More details on training and setup of baselines can be found in the supplementary material. During testing, we apply the trained model on all testing graph pairs. For medium-size synthetic testing graph pairs, each method is given a budget of 500 search iterations. For large real-world graph pairs, each method is given a budget of 7500 search iterations. These budgets were chosen based on when the models' performances stabilized. Details about performance using other iteration budgets may be found in the Supplementary Material.

To validate the usefulness of the learned DQN, we compare GLSEARCH, our full model, with a randomly initialized model, GLSEARCH-RAND, which replaces the output of our DQN with a completely random scalar. We show the performance gain of our model through training by substantially outperforming this baseline on all real-world datasets.

4.2 PARAMETER SETTINGS

For I-PCA, NEURALMCS and GLSEARCH, we utilize 3 layers of Graph Attention Networks (GAT) (Velickovic et al., 2018) each with 64 dimensions for the embeddings. The initial node embedding is encoded using the local degree profile (Cai & Wang, 2018). We use $\mathrm{ELU}(x) = \alpha(\exp(x)–1)$ for $x \leq 0$ and x for $x > 0$ as our activation function where $\alpha = 1$. We run all experiments with Intel i7-6800K CPU and one Nvidia Titan GPU. For DQN, we use MLP layers to project concatenated embeddings to a scalar. We use SUM followed by an MLP for READOUT and 1DCONV+MAXPOOL followed by an MLP for INTERACT. For training, we set the learning rate to 0.001, the number of training iterations to 10000, and use the Adam optimizer (Kingma & Ba, 2015). The models were implemented with the PyTorch and PyTorch Geometric libraries (Fey & Lenssen, 2019).

4.3 RESULTS

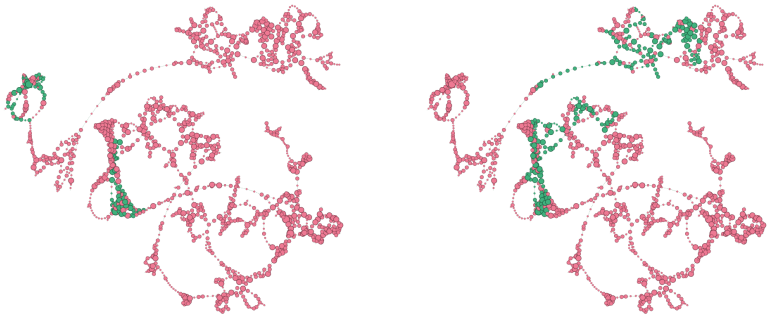
The key property of GLSEARCH is its ability to find the best solution in the fewest number of search iterations. As shown in Table 1, our model outperforms baselines in terms of size of extracted subgraphs on all medium-sized synthetic graph datasets. However, baseline solvers are already quite powerful on these datasets. As it is easy to extract the maximum common subgraph on smaller graph datasets because the total search space grows exponentially with graph size, to truly show the

Table 2: Results on real-world large graph pairs. Each dataset consists of one large real graph pair ($\mathcal{G}_1$, $\mathcal{G}_2$ may not be isomorphic, but $\mathcal{G}_{1s}$, $\mathcal{G}_{2s}$ are isomorphic guaranteed by search). Below each dataset name, we show its size $\min(|\mathcal{V}_1|, |\mathcal{V}_2|)$ to indicate these pairs are significantly larger than the ones in Table 1. Consistent with Table 1, we show the ratio of the subgraph sizes.

Method	**ROAD** 652	**DBEN** 1945	**DBZH** 1907	**DBPD** 1907	**ENRO** 3369	**COPR** 3518	**CIRC** 4275	**HPPI** 2152
MCSP	0.374	0.815	0.797	0.722	0.694	0.684	0.498	0.864
MCSP+RL	0.771	0.699	0.589	0.434	0.742	0.674	0.583	0.787
GW-QAP	0.305	0.929	0.855	0.808	0.711	0.860	0.354	0.834
I-PCA	0.267	0.551	0.589	0.607	0.650	0.707	0.203	0.762
NEURALMCS	0.977	0.785	0.616	0.620	0.737	0.742	0.561	0.785
GLSEARCH-RAND	0.641	0.762	0.658	0.639	0.814	0.755	0.603	0.814
GLSEARCH	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
BEST SOLUTION SIZE	131	508	482	521	543	791	3515	404

performance advantage of GLSEARCH, we also run experiments on large real-world graph datasets with thousands of nodes.

As shown in Table 2, our model outperforms baselines in terms of the size of the extracted subgraphs on all large real-world datasets. The exact solvers rely on heuristics for node selection, and consistently find smaller subgraphs compared to our results. Figure 3 compares results by MCSP and GLSEARCH. Since MCSP selects nodes with large degrees as its heuristic, the selected nodes tend to be confined in one dense cluster of large degree nodes in $\mathcal{G}_1$. This implies the subgraph in $\mathcal{G}_2$ matched to this dense cluster must also be dense (isomorphism constraint of MCS). In contrast, GLSEARCH is able to find long chains in $\mathcal{G}_1$ which allows easier matching in $\mathcal{G}_2$. In general, there are many cases of large real-world graph pairs where heuristics are not enough to extract large high quality subgraphs. Due to its leveraging both learning and search, GLSEARCH consistently finds subgraphs more than double the size of those found by search based baselines for large real-world graph pairs.



(a) MCSP (b) GLSEARCH

Figure 3: Visualization of MCS results on ROAD. Nodes with large degrees have large circles. For each method, we show the two graphs being matched. Selected subgraphs are colored in green.

Compared with learning based graph matching models, GLSEARCH is the only model which learns a reward that is dependent on both state and action, i.e. $Q(s_t, a_t)$. GW-QAP, I-PCA, and NEURALMCS essentially pre-compute the matching scores for all the node pairs in the input graphs, and therefore at each search step, the scores cannot adapt to the particular state, i.e. the matching scores only depend on $\mathcal{G}_1, \mathcal{G}_2$. Notice our state representation includes $\mathcal{G}_1, \mathcal{G}_2$ as well, hence GLSEARCH has more representational power than baselines. Trained under a reinforcement learning framework guided by search, GLSEARCH also performs the best among learning based baselines.

4.4 ABLATION AND PARAMETER STUDY

To evaluate the effectiveness of different components proposed in our DQN model, we run ablation studies on all real world datasets.

We first measure the importance of each embedding vector fed to our DQN module, as described by Equation 3. We remove each embedding vector (specifically: $\boldsymbol{h}_{\mathcal{G}} = \text{INTERACT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2})$,

Table 3: Abaltion study on real datasets.

Method	ROAD	DBEN	DBZH	DBPD	ENRO	COPR	CIRC	HPPI
GLSEARCH (no $\boldsymbol{h}_{\mathcal{G}}$)	0.977	0.878	0.925	0.845	0.860	0.987	0.980	0.960
GLSEARCH (no $\boldsymbol{h}_s$)	**1.000**	0.874	0.894	0.869	0.928	**1.000**	0.801	0.913
GLSEARCH (no $\boldsymbol{h}_{\mathcal{D}c}$)	0.803	0.780	0.687	0.818	0.740	0.804	0.505	0.849
GLSEARCH (no $\boldsymbol{h}_{\mathcal{D}0}$)	0.576	0.856	0.782	0.768	0.823	0.932	0.323	0.938
GLSEARCH (SUM interact)	0.902	0.913	0.963	0.885	0.899	0.957	**1.000**	0.948
GLSEARCH (unfactored)	0.447	0.807	0.712	0.582	0.816	0.816	0.512	0.861
GLSEARCH (unfactored-i)	0.500	0.789	0.741	0.772	0.748	0.825	0.902	0.864
GLSEARCH	0.992	**1.000**	**1.000**	**1.000**	**1.000**	0.990	0.881	**1.000**
BEST SOLUTION SIZE	132	508	482	521	543	799	3989	404

$\boldsymbol{h}_s = \text{INTERACT}(\boldsymbol{h}_{s1}, \boldsymbol{h}_{s2})$, $\boldsymbol{h}_{\mathcal{D}c}$, and $\boldsymbol{h}_{\mathcal{D}0}$) individually from the DQN model and retrain the model under the same training settings. Table 3 is consistent with our conclusion that every embedding vector used by GLSEARCH is critical in capturing the search state's representation. Furthermore, we find leveraging bidomain representations is very beneficial to our model.

We next measure the importance of interaction to address the graph symmetry issue, where input graph pairs must be order insensitive. We first test the necessity of using more complex interaction functions, by replacing our 1DCONV+MAXPOOL interaction with simple SUM for interaction (still followed by an MLP). As shown in Table 3, we see that simpler interaction functions may not be powerful enough to encode the interaction between 2 graphs. Particularly, this suggests that interaction is quite important to model performance.

Finally, we measure the importance of factoring out actions from our DQN model. We test this with 2 models. The first utilizes Equation 1 to encode the Q-value, which we refer to as GLSEARCH (unfactored). Since Equation 1 also suffers from the issue of graph symmetry, we adapt this model to use the same interaction function as GLSEARCH to construct 3 order-invariant embeddings $\boldsymbol{h}_{\mathcal{G}} = \text{INTERACT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2})$, $h_s = \text{INTERACT}(\boldsymbol{h}_{s1}, \boldsymbol{h}_{s2})$, $\boldsymbol{h}_a = \text{INTERACT}(\boldsymbol{h}_i, \boldsymbol{h}_j)$ to concatenate and pass to the final MLP layer in Equation 1. We refer to this model as GLSEARCH (unfactored-i). Our results show that without factoring out the action, our performance is comparable to or worse than MCSP, indicating the significant performance boost introduced by maximally reflecting the effect of adding node pairs.

5 RELATED WORK

MCS detection is NP-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. Closely 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., 2019; 2020a; Li et al., 2019; Ling et al., 2020), or rely on labeled instances (Bai et al., 2020b).

6 CONCLUSION

We believe the interaction of search and learning is a promising direction for future research, and take a step towards bridging the gap by tackling the NP-hard challenging task, Maximum Common Subgraph detection. We have proposed a reinforcement learning method which unifies search and deep Q-learning into a single framework. By using the search to train our carefully designed DQN, the DQN provides better node selection policy for search to find large common subgraph solutions faster, which is experimentally verified on real-world large graph pairs. In future, the adaptation of our framework to other NP-hard tasks requiring search can be explored.

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*, 2019.

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.

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.

Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018.

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.

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.

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

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.

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

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

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.

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.

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.

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.

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

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.