GLSearch: Maximum Common Subgraph Detection via Learning to Search

Yunsheng Bai [*1] **Derek Xu** [*1] **Yizhou Sun** [1] **Wei Wang** [1]

Abstract

Detecting the Maximum Common Subgraph (MCS) between two input graphs is fundamental for applications in drug synthesis, malware detection, cloud computing, etc. However, MCS computation is NP-hard, and state-of-the-art MCS solvers rely on heuristic search algorithms which in practice cannot find good solution for large graph pairs given a limited computation budget. We propose GLSEARCH, a Graph Neural Network (GNN) based *learning to search* model. Our model is built upon the branch and bound algorithm, which selects one pair of nodes from the two input graphs to expand at a time. We propose a novel GNN-based Deep Q-Network (DQN) to select the node pair, making the search process much faster. Experiments on synthetic and real-world graph pairs demonstrate that our model learns a search strategy that is able to detect significantly larger common subgraphs than existing MCS solvers given the same computation budget. GLSEARCH can be potentially extended to solve many other combinatorial problems with constraints on graphs.

1. Introduction

Graphs gain increasing attention recently due to their expressive nature in representing real-world data and recent successes in addressing challenging graph tasks via learning, represented by graph neural networks. 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 broad utilities in many domains such as software analysis (Park et al., 2013), graph

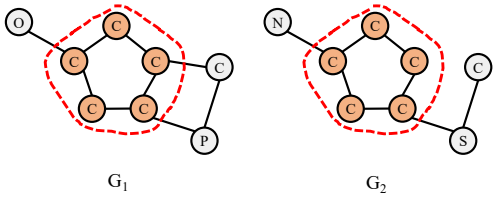


Figure 1: For graph pair $(\mathcal{G}_1, \mathcal{G}_2)$ with node labels, the induced connected Maximum Common Subgraph (MCS) is the five-member ring structure highlighted in circle.

database systems (Yan et al., 2005) and cloud computing platforms (Cao et al., 2011). For example, in drug synthesis, finding similar substructures in compounds with similar properties can reduce manual labor (Ehrlich & Rarey, 2011).

MCS detection is NP-hard in its nature and is thus very challenging. The state-of-the-art exact MCS detection algorithms, which use a powerful branch and bound search framework, still run in exponential time in the worst case (Liu et al., 2019). These algorithms aim to provably extract the MCS by exhausting the search space as efficiently as possible. However, in large real-world graphs, exhausting the search space is not computationally tractable. 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 low-degree nodes.

Recently, there are some related efforts from the learning community; however, these methods fall short in tackling the constraint posed by the MCS definition that the two extracted subgraphs must be isomorphic to each other. For example, Wang et al. (2019) aims to detect a soft matching matrix between nodes in two input graphs, which, however, cannot be easily transformed into the discrete matched subgraph. Bai et al. (2020b) is the first attempt to use learning based approach to directly output MCS. However, it heavily relies on labeled MCS instances, which requires pre-computation of MCS results by running exact solvers.

In this paper, we present GLSEARCH (*G*raph *L*earning to *S*earch), a general framework for MCS detection combining the advantages of search and deep Reinforcement Learning (RL). GLSEARCH learns to search by adopting a Deep

[*]Equal contribution [1]Department of Computer Science, University of California, Los Angeles, California, USA. Correspondence to: Yunsheng Bai <yba@ucla.edu>, Derek Xu <derekqxu@ucla.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Q-Network (DQN) (Mnih et al., 2015) to replace the node selection heuristics required in state-of-the-art MCS solvers, leading to faster arrival of the optimal solution for an input graph pair, which is particularly useful when applied to large real-world graphs and/or with a limited search budget. Our method proposes a novel representation for states in the DQN framework, and reformulates DQN in a novel way to better capture the effect of different node selections, exploiting the representational power of Graph Neural Networks (GNN).

Besides the novel state representation, another key component of GLSEARCH is the search algorithm. Since the search space is very large especially for large graphs pairs, and DQN policy may choose a bad action during search, the traditional DQN framework tends to stop searching too early, leading to suboptimal solutions. To address this, we adopt a branch-and-bound search framework with backtracking.

Lastly, 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 large real graph datasets (that are significantly larger than the 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 important yet challenging task of Maximum Common Subgraph detection for general-domain input graph pairs and propose GLSEARCH as the solution.
- The key novelty is of GLSEARCH is its learning to search framework. With a DQN reformulation trick, 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 DQN training in a pre-training stage and an imitation learning stage.
- We conduct extensive experiments on medium, large, and million-node real-world graphs to demonstrate the effectiveness of the proposed approach compared against a series of strong baselines in MCS detection and graph matching.

2. Preliminaries and Related Work

2.1. The MCS Detection Problem

We denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ denote the vertex and edge set. A node-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 $(i, j) \in \mathcal{E}$. In this paper, we aim to detect 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 node-induced subgraph 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 $\mathcal{G}_1$ (or $\mathcal{G}_2$) is subgraph isomorphic to $\mathcal{G}_2$ (or $\mathcal{G}_1$).

2.2. Related Work

Traditional Efforts 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), integer programming (Bahiense et al., 2012), conversion to maximum clique detection (Levi, 1973; McCreesh et al., 2016), etc., among which MCSP+RL (Liu et al., 2019) (details presented in Section 2.3) is the state-of-the-art method, which guarantees to find common subgraphs satisfying the isomorphism constraint, but usually cannot extract large common subgraphs when input graphs become large.

Efforts on Learning to Solve Graph Similarity and Matching There is a growing trend of using machine learning approaches to graph matching and similarity score computation (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; Liu et al., 2020; Chen et al., 2020). These methods cannot handle the isomorphism constraint in MCS well, since they were mainly designed for tasks without hard constraints, e.g. finding the similarity score or node-node matching between two graphs supervised by true similarity or matching. Thus, to better satisfy the constraints of MCS, these models need to be embedded into a search framework that uses the scores provided by the models to guide the search for MCS, which will be described next. For example, GW-QAP performs Gromov-Wasserstein discrepancy (Peyré et al., 2016) based optimization and outputs a matching matrix for all node pairs indicating the likelihood of matching (Zanfir & Sminchisescu, 2018). I-PCA performs image matching by outputting a doubly-stochastic matching matrix computed from intermediary Convolution Neural Network features from an input image pair (Wang et al., 2019). Some other methods tackle tasks with hard constraints, such as MCS or GED, but these methods require supervision, which is difficult to obtain (Bai et al., 2020b; Wang et al., 2020). A recent work proposes to use RL to overcome the supervision limitation; however, this approach relies heavily on existing heuristics

to work well (Liu et al., 2019).

Efforts on Learning to Solve NP-hard Graph Problems Existing works such as Dai et al. (2017), Gasse et al. (2019), and Fan et al. (2020) focus on designing learning based approaches for solving NP-hard tasks on graphs, e.g. Minimum Vertex Cover, Mixed Integer Linear Programming, Network Dismantling, etc., but our problem, Maximum Common Subgraph detection, operates on a pair of input graphs instead of a single graph. Besides, MCS detection requires hard constraint satisfaction, i.e. isomorphism of extracted subgraphs, which is handled by a search algorithm described next.

2.3. Search Algorithms for MCS

In this section, we present the state-of-the-art branch and bound search framework for detecting MCS as shown in Algorithm 1 and Figure 2, which allows the exploration of search space and guarantees the satisfaction of the isomorphism constraint posed by MCS. Thus, it serves as the backbone of our proposed approach. We then discuss several drawbacks in the existing search-based MCS detection algorithms.

Algorithm 1 Branch and Bound for MCS. We highlight in green boxes the two places that will be replaced by GLSEARCH.

1: **Input:** Input graph pair $\mathcal{G}_1, \mathcal{G}_2$.
2: **Output:** $maxSol$.
3: Initialize $s_0 \leftarrow$ empty state.
4: Initialize stack $\leftarrow$ `new` Stack(s_0).
5: Initialize $maxSol \leftarrow$ empty solution.
6: **while** stack $\neq \emptyset$ **do**
7: $s_t \leftarrow$ stack.$pop()$;
8: $curSol \leftarrow s_t.getCurSol()$;
9: **if** $|curSol| > |maxSol|$ **then**
10: $maxSol \leftarrow curSol$;
11: **end if**
12: $UB_t \leftarrow |curSol|$ + overestimate(s_t);
13: **if** $UB_t \leq |maxSol|$ **or** $|s_t.\text{actions}| = 0$ **then**
14: **continue**;
15: **end if**
16: $\mathcal{A}_t \leftarrow s_t$.actions;
17: $a_t \leftarrow policy(s_t, \mathcal{A}_t)$;
18: s_t.actions $\leftarrow s_t$.actions $\setminus \{a_t\}$;
19: stack.$push(s_t)$;
20: $s_{t+1} \leftarrow$ environment.update($s_t, \mathcal{A}_t$);
21: stack.$push(s_{t+1})$;
22: **end while**

MCSP and Its Limitations 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, starting from an empty subgraph, grows the matching subgraph one node pair (between the two graphs) at a time and maintains the best solution found so far. In each search iteration, denote the current search state as s_t consisting of $\mathcal{G}_1$, $\mathcal{G}_2$, the current matched 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, (i, j) added to the currently selected subgraphs, where node i is from $\mathcal{G}_1$ and node j is from $\mathcal{G}_2$, as its action, denoted as a_t. It then decides to either continue the search if the solution is promising, or otherwise backtrack to the parent search state, i.e. the current search state is pruned (line 14). Various heuristics on node pair selection policy, denoted as "$policy$" in line 17, are proposed in MCSP and MCSP+RL. For example, in MCSP, nodes of large degrees are selected before small-degree nodes.

At each search state, in order to determine whether the solution is promising or not, an upper bound of the size of the MCS, "UB_t" in line 12 is computed. A concept of "bidomain" is introduced to facilitate its estimation. Bidomains partition the unselected nodes, $(V_1 \setminus V_{1s}) \cup (V_2 \setminus V_{2s})$, into equivalent classes, $\mathcal{D} = \{D_k\}_{k=0}^{2^{|V_{1s}|}-1}$, where the k-th bidomain D_k consists of two sets of nodes, $D_k = \langle \mathcal{V}'_{k1}, \mathcal{V}'_{k2} \rangle$ such that nodes in $\mathcal{V}'_{k1}$ and nodes in $\mathcal{V}'_{k2}$ connect to subgraph nodes, in $\mathcal{V}_{1s}$ and $\mathcal{V}_{2s}$, that match to each other. Figure 3 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. Since we require the MCS to be connected subgraphs, we differentiate bidomains $\mathcal{D}^{(c)} = \{\mathcal{D}_k\}_{k=1}^{2^{|V_{1s}|}-1}$ that are connected (adjacent) to $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ (e.g. bidomain "01" and "10" in Figure 3) from the single bidomain D_0 disconnected (unconnected) from $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$ (e.g. bidomain "00" in Figure 3) The candidate node pairs to select from, i.e. the action space "$\mathcal{A}_t$", consists of all node pairs in all connected bidomains, $\mathcal{D}^{(c)}$. This also guarantees the extracted subgraphs at each state are isomorphic to each other.

To estimate the upper bound, it is noteworthy that each bidomain can contribute at most $\min(|\mathcal{V}'_{k1}|, |\mathcal{V}'_{k2}|)$ nodes to the future best solution. The upper bound can therefore be estimated as $\sum_{D_k \in \mathcal{D}} \min(|\mathcal{V}'_{k1}|, |\mathcal{V}'_{k2}|)$, which is the "overestimate(s_t)" function in line 12. This upper bound computation is consistently used for all the methods in the paper. The major difference is and the node pair selection policy and backtracking method, i.e. lines 17 and 7.

As mentioned previously, MCSP adopts a heuristic that selects node pairs with the largest degree as its policy. The most severe limitation of MCSP is that the node-degree-

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

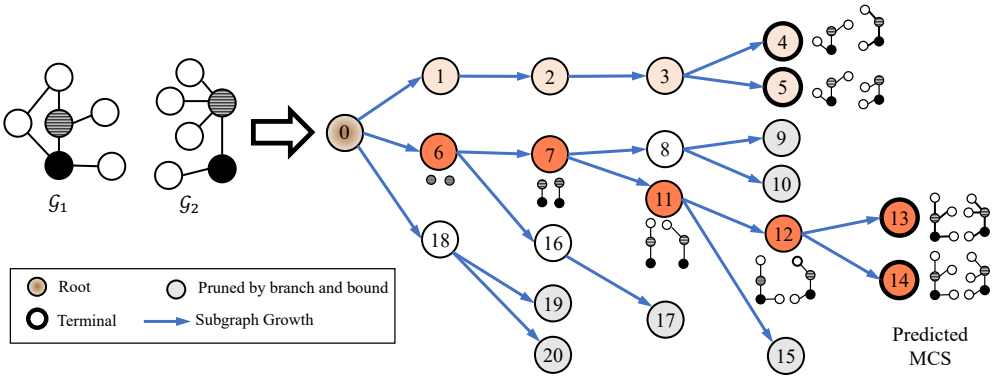


Figure 2: An illustration of the branch and bound search algorithm for MCS detection (Algorithm 1). Each node represents one state (s_t), with node id reflecting the order states are visited, and each edge represents an action (a_t) of selecting a new node pair. The search is essentially depth-first with pruning by the upper bound check. Our model learns the node pair selection strategy, i.e. which state to visit first. Notice, if the policy had visited state 6 before state 1, a large solution may be found in less iterations. When the search completes or a pre-defined search iteration budget is used up, the best solution (largest subgraphs) will be returned, which in this example is state 13 (and 14).

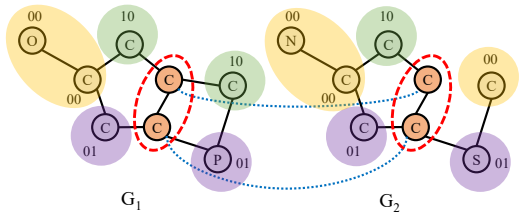


Figure 3: An example to illustrate the concept of bidomains. We denote the bidomain of each non-subgraph node by its bitstring, where each bit denotes whether there is a connection ("1") or not ("0") to a particularly node in subgraph $\mathcal{V}_{1s}$ or the node it maps to in subgraph $\mathcal{V}_{2s}$ (both circled in red). For example, each node in the "10" bidomain is connected to the top "C" node in the subgraph and disconnected to the bottom "C" node. In this case, there are 3 bidomains denoted as "00" (D_0), "01" (D_1), and "10" (D_2). By definition, the bidomain denoted with all zeros, e.g. "00" in this case, is called the disconnected bidomain. Notice, the bidomain corresponding to "11" (D_3) is empty.

based heuristic is not adaptive to the complex real-world graph structures. McSp always backtracks to the parent search state; however, this leads to suboptimal solutions, as the wrong action could have occured earlier in search.

McSp+RL and Its Limitations McSp+RL improves McSp by replacing the node pair selection policy with a value function for each node or node pair. Their goal is to minimize the search tree size so that a search tree leaf can be reached as early as possible. Specifically, McSp+RL aims to reduce the UB_t to make it tighter, so that more pruning (line 14) can happen in subsequent search steps, resulting in a smaller search tree (search space). To achieve that goal, they design the reward function for each state-action pair as the reduction (or reduction rate) of search space by selecting that node pair. The value function maintains a score for each node (or node pair), which is initialized to 0 and updated during search. In each step during search, the policy is to select the node pair with the largest score.

We identify two limitations of McSp+RL: (1) Since the reward definition is defined using a heuristic and there is no training stage, for each new graph pair, the scores must be re-initialized and the policy has to be re-updated. (2) The fact that the scores for each node (or node pair) are 0 at the beginning of search has another problem: McSp+RL breaks ties using node degrees, essentially degenerating to the same policy as McSp initially for each graph pair, which is also verified by our experiments as shown in Figure 4 where McSp and McSp+RL perform the same.

Besides, a common issue of McSp and McSp+RL is that, during the search, they can enter a bad locally optimal search state and get "stuck" without finding a better (larger) solution, $maxSol$, for many iterations, as shown in the flat line segments of Figure 4.

3. Proposed Method

In this section, we present our RL based MCS detection method, GLSearch. The rest of Section 3 is organized as follows. Section 3.1 presents a high-level overview of how to leverage Deep Q-Network (DQN) (Mnih et al., 2015) for search, including the basic definitions of state, action, reward, etc., and how DQN can address the various issues of search methods for MCS described previously. Section 3.2 describes the details of how to leverage DQN for search, focusing on how to effectively design representation learning for DQN for the task of MCS detection. Section 3.3 explains how to effectively train the DQN with the help of search, i.e. how search can in turn help DQN training.

Table 1: Table of Notations.

Notation	Description
$\mathcal{G}_1, \mathcal{G}_2$	The input pair of graphs.
$\mathcal{G}_{1s}, \mathcal{G}_{2s}$	The selected subgraph of some given state.
$\mathcal{V}_x, \mathcal{E}_x$	Set of nodes and edges for graph $\mathcal{G}_x$.
s_t	The search state at iteration t.
a_t	The action selected at iteration t.
$\mathcal{D}$	Set of all bidomains, $\{D_k\}_{k=0}^{2^{\lvert V_{1s}\rvert}-1}$.
$\mathcal{D}^{(c)}$	Set of connected bidomains, $\{\mathcal{D}_k\}_{k=1}^{2^{\lvert V_{1s}\rvert}-1}$.
D_0	The disconnected bidomain.
$\mathcal{V}'_{k1}, \mathcal{V}'_{k2}$	Set of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$ in D_k.
h_i, h_j	Embeddings of nodes $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$.
h_{s1}, h_{s2}	Embeddings of subgraphs $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$.
$h_{\mathcal{G}_1}, h_{\mathcal{G}_1}$	Embeddings of graph $\mathcal{G}_1$ and subgraph $\mathcal{G}_2$.
$h_{D^{(c)}}$	Embedding for connected bidomains, $D^{(c)}$
h_{D_0}	Embedding for disconnected bidomains, D_0

3.1. Leveraging DQN for Search: Overview

GLSEARCH enables graph representation learning techniques to tackle the hard isomorphism constraint posed by MCS and uses deep Q-learning to select node pairs smartly in each search state. GLSEARCH represents states and actions in continuous embeddings, and maps (s_t, a_t) to a score $Q(s_t, a_t)$ via a DQN which consists of a Graph Neural Network encoder and learnable components to project the representations into the final score. GLSEARCH is trained on a set of diverse small and medium-sized graphs, and once trained, can be applied to any new graph pair.

Unlike MCSP and MCSP+RL, which aim to reduce the search tree size, the aim of our agent is to directly maximize the common subgraph size, allowing large common subgraph to be found even on very large graph pairs.

State s_t consists of the (1) current selected subgraphs, (2) the node-node mappings between the nodes in the selected subgraphs, and (3) the input graphs. We include the node-node mappings as part of the state definition since node-node mappings can be used to derive the bidomain partitioning as illustrated in Figure 3, which constrains the node pairs that can be selected in future, and thus affects the future common subgraph size. Action a_t is defined as a node pair to select. For GLSEARCH, given our goal, the immediate reward for transitioning from one state to any next state is defined as $r_t = +1$ since one new node pair is selected, so that $Q(s_t, a_t)$ captures the largest common subgraph size starting at s_t by performing a_t.

GLSEARCH is trained to find large common subgraphs quickly, but due to the large action space of large graph pairs, our model may still be susceptible to the local optimum without increasing $maxSol$ as described in Section 2.3. Thus, when this occurs, we utilize additional information stored in the search tree to backtrack to a state that will most likely improve $maxSol$. We find that in practice, states with a large action space, $\mathcal{A}_t$, tend to include more high-quality unexplored actions. Hence, if the best solution found so far does not increase[2] for a pre-defined number of iterations, then in the next iteration, instead of popping from the stack[3], we find the state with the largest action space, and visit it. We refer to this improved search methodology as **promise-based search**. More details can be found in the supplementary material.

3.2. Search Policy Learning via GNN-based 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 remaining future reward after selecting action $a_t = (i, j)$ where $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$, which intuitively corresponds to the largest number of nodes that will be eventually selected starting from the action edge (s_t, a_t) as shown 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 $\boldsymbol{h}_{s2}$, 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 would then be designed as:

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

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 $Q(s_t, a_t)$ should capture the *long-term* effect of adding (i, j). What is worse, different node pairs have different embeddings, but their immediate rewards are always $+1$, a constant, in MCS, making differentiating the quality of different actions even more difficult.

(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.

[2] If the search does not enter line 10 of Algorithm 1.

[3] Line 7 of Algorithm 1.

(C) Lastly, how to effectively leverage the node-node mappings between $\mathcal{G}_{1s}$ and $\mathcal{G}_{2s}$, an important part of the state definition as explained in Section 3.1, 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 reformulate the optimal Q score, $Q^*(s_t, a_t)$, as $r_t + \gamma V^*(s_{t+1}) = 1 + \gamma V^*(s_{t+1})$ (using the fact that $r_t = +1$) in MCS, where V is the value function, 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 $\boldsymbol{h}_j$. In this case, we can rely on our state embedding to capture global information and amplify differences between different actions by looking at the states they will arrive.

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 $\text{INTERACT}(\cdot)$ is any commutative function to combine the two embeddings. Instead of summation, We define a CNN-based $\text{INTERACT}(\cdot)$ operator, defined in Section B.3 of the Supplementary Material, since cross-graph interaction is important for the MCS task. This interacted embedding is later concatenated with other useful representations and fed into a final MLP to compute the Q score.

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 2.3, bidomains have been adopted 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 also use the representation of bidomains to amplify the differences in different states. Denote $\boldsymbol{h}_{D_k}$ as the representation for bidomain $D_k = \langle \mathcal{V}'_{k1}, \mathcal{V}'_{k2} \rangle$. Similar to computing the graph-level and subgraph-level embeddings, we compute $\boldsymbol{h}_{D_k}$ as

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

Given all the bidomain embeddings, we compute a single representation for all the connected bidomains, $\mathcal{D}^{(c)}$, $\boldsymbol{h}_{\mathcal{D}c} = \text{READOUT}(\{\boldsymbol{h}_{D_k} | k \in \mathcal{D}^{(c)}\})$. Notice, since bidomains are a function of node-node mappings, the mapping is implicitly accounted for by this representation. In fact, bidomain representations are better at capturing difference in states than node-node mappings. Across different actions, the node-node mappings only differ by a single node-node pair, but the bidomains can differ drastically as they capture the effect of matching certain nodes, rather than the mappings themselves. Our final DQN has the form:

$$Q(s_t, a_t) = 1 + \gamma \text{MLP}\Big(\text{CONCAT}\big(\text{INTERACT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2}), \text{INTERACT}(\boldsymbol{h}_{s1}, \boldsymbol{h}_{s2}), \boldsymbol{h}_{\mathcal{D}c}, \boldsymbol{h}_{D_0}\big)\Big). \tag{3}$$

3.3. Leveraging Search for DQN Training

For large graph pairs, the action space can be quite large. 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 run the search to completion on small graph pairs (thus, the exact MCS solution is found), and use a supervised mse loss function to replace the DQN loss function. The overall loss function 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 score. In this case, y_t denotes the remaining size of the largest common subgraph starting from s_t to its leaf node in the current branch of 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 follow the expert trajectories provided by McSp instead of its own predicted $Q(s_t, a_t)$, to incorporate more trustworthy policy decisions into the training signal. 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 graph matching methods from various domains. We conduct experiments on 7 hundred-node medium-sized synthetic and real-world graph datasets, 8 thousand-node large real-world graph datasets, and 2 million-node very large real-world 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 cannot 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.

4.1. Baseline Methods

There are two groups of methods: Exact MCS algorithms including McSp and McSp+RL, learning based graph matching models including GW-QAP, I-PCA, and NeuralMcs.

Table 2: Results on medium graphs. Each synthetic 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. NCI109 consists of 100 chemical compound graph pairs whose average graph size is 28.73. 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**	**NCI109**
MCSP	0.913	0.892	0.842	0.896	0.905	0.856	0.948
MCSP+RL	0.923	0.857	0.844	0.877	0.913	0.875	0.948
GW-QAP	0.945	0.887	0.855	0.925	0.916	0.898	0.966
I-PCA	0.899	0.863	0.848	0.923	0.879	0.852	0.951
NEURALMCS	0.908	0.889	0.846	0.906	0.889	0.865	0.954
GLSEARCH-RAND	0.995	0.987	0.920	0.978	0.967	0.931	0.989
GLSEARCH	**1.000**	**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	10.48

Table 3: 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 2. Consistent with Table 2, 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

All the methods either originally use or are adapted to use the branch and bound search framework in Section 2.3 with differences in node pair selection policy and training strategies. During testing, we apply the trained model on all testing graph pairs. We give a budget of 500 and 7500 search iterations for medium-size and large graph pairs. For each of the two million-node graph pairs, since the true MCS is much larger, we run each method for 50 minutes and plot the subgraph size growth across time. Due to the search algorithm, all the methods can find exact MCS solutions given long enough budget, albeit an unrealistic assumption in practice for large graph pairs.

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.

4.2. Hyperparameter Settings

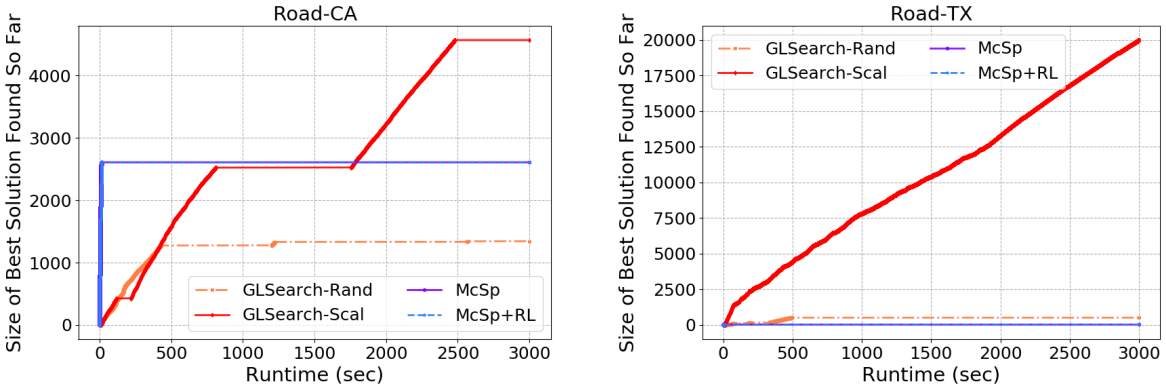
For I-PCA, NEURALMCS and GLSEARCH, we utilize 3 layers of Graph Attention Networks (GAT) (Velickovic et al., 2018) each with 64 dimensions for the embeddings. The initial node embedding is encoded using the local degree profile (Cai & Wang, 2018). We use $\mathrm{ELU}(x) = \alpha(\exp(x)–1)$ for $x \leq 0$ 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. Further details can be found in supplementary material. 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 2, our model outperforms baselines in terms of size of extracted subgraphs on all medium-sized synthetic graph datasets and the chemical compound dataset NCI109.

Regarding large real-world graphs, as shown in Table 3, our model outperforms baselines in terms of the size of the extracted subgraphs on all datasets. The exact solvers rely on heuristics for node selection, and consistently find much smaller subgraphs compared to our results.

Compared with learning based graph matching models,



(a) Result on ROAD-CA with 978513 nodes.

(b) Result on ROAD-TX with 1080909 nodes.

Figure 4: Comparison of the best solution sizes of different methods on two million-node graph pairs, ROAD-CA and ROAD-TX. GW-QAP, I-PCA, and NEURALMCS encounter memory error on these graph pairs due to their computation of a quadratic node-node matching matrix

Table 4: Ablation study on large real-world datasets. We demonstrate our Q function design choices indeed solve the various shortcomings presented in Section 3.2, through better representation learning.

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

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. Million-Node Graph Pairs

GLSEARCH can scale to very large graph pairs, the limit of which is only bounded by the scalability of the GNN embedding step. To demonstrate this, we run GLSEARCH on million-node real-world graph datasets, ROAD-CA and ROAD-TX. To fit the model onto our GPU resources, we construct a lighter version of GLSEARCH, called GLSEARCH-SCAL, which reduces the GAT encoder dimensions from 64 to 16.

As shown in Figure 4, GLSEARCH significantly outperforms baseline solvers on the two million-node real-world datasets. On ROAD-TX, MCSP and MCSP+RL perform poorly (getting "stuck" in local optimum as pointed out in Section 2.3) while GLSEARCH continues to find larger and larger common subgraph after 50 minutes.

4.5. Ablation Study

To evaluate the effectiveness of different components proposed in our DQN model, we run ablation studies on the 8 large 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: $\boldsymbol{h}_{\mathcal{G}} = \text{INTERACT}(\boldsymbol{h}_{\mathcal{G}_1}, \boldsymbol{h}_{\mathcal{G}_2})$, $\boldsymbol{h}_s = \text{INTERACT}(\boldsymbol{h}_{s1}, \boldsymbol{h}_{s2})$, $\boldsymbol{h}_{\mathcal{D}c}$, and $\boldsymbol{h}_{D_0}$) individually from the DQN model and retrain the model under the same training settings. Table 4 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 symmetry issue of the MCS calculation, 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 4, 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 factoring out the action.

5. Insights and Contributions of GLSEARCH

To Search community on MCS detection The major challenge that prevents existing search algorithms from extracting large common subgraphs for large input graph pairs is that the focus of these algorithms is on reducing the **overall** search space rather than making smarter node pair selections in **each** search step. By improving the order it searches candidate solutions, GLSEARCH can quickly find ***better*** MCS candidates, without much (or any) backtracking and pruning, than state-of-the-art search algorithms.

To General Learning community GLSEARCH tackles the hard constraint that the subgraphs must be isomorphic to each other by only choosing actions from connected bidomains. However, there is an additional advantage of introducing the bidomain concept: Bidomains partition the rest of the input graphs into different regions where future actions can be selected. Thus, properly encoding of the bidomains gives more information about hard constraints to the DQN, which improves performance as experimentally verified by the ablation studies. More generally, this shows that learning components can be further enriched by incorporating knowledge on tackling hard constraints of an NP-hard task (e.g. bidomain) into their model design.

To Graph Deep Learning community Although various works have pointed out and analyzed the limitation of GNN's expressive power (Xu et al., 2019c; Balcilar et al., 2021), for particular tasks such as MCS detection, GNNs can still be used if augmented properly. We aim to predict a Q score for a state-action pair, using a DQN with two components, one component computing the local node embeddings using several layers of GNN, the other component combining embeddings at a different granularity, i.e. embeddings at the subgraph, whole-graph, and bidomain levels, to produce the final score. Overall our DQN design adopts a similar general principle as Position-Aware GNN (You et al., 2019), which allows a regular GNN to absorb information from non-local nodes (called "anchor" nodes which are randomly selected nodes globally).

To Reinforcement Learning community We are aware of efforts in the RL community to tackle NP-hard problems, but they either focus on non-graph tasks, such as KnapSack (Bello et al., 2017) and Job Shop Scheduling (Solozabal et al., 2020; Dai et al., 2019), or address single-graph NP-hard tasks without hard constraints, such as Minimum Vertex Cover (Dai et al., 2017), Graph Exploration (Dai et al., 2019), and Network Dismantling (Fan et al., 2020). Fundamentally different from these works, MCS detection requires a graph pair as input, and we show how to properly encode such an input. The subgraph isomorphism constraint of the task also sets us apart from the aforementioned graph tasks which we tackle via fully taking advantage of a key property of the task, i.e. bidomain, while in contrast, Solozabal et al. (2020) relies on penalty signals generated from constraint dissatisfaction in order to guide the agent to achieve feasible solutions for non-graph tasks.

6. Conclusion

We believe the interplay of search and learning is a promising research direction, 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 synthetic and real-world large graph pairs. In the future, we will explore the adaptation of our framework which combines learning with search to other constrained combinatorial problems, e.g. Maximum Clique Detection.

7. Acknowledgements

This work was partially supported by NSF IIS- 2031187, NSF DGE-1829071, NSF III-1705169, NSF CAREER Award 1741634, NSF 1937599, NIH R35-HL135772, NIH/NIBIB R01 EB027650, DARPA HR00112090027, Okawa Foundation Grant, and Amazon Research Awards.

References

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

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., and Wang, W. Simgnn: A neural network approach to fast graph similarity computation. *WSDM*, 2019.

Bai, Y., Ding, H., Gu, K., , Sun, Y., and Wang, W. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. *AAAI*, 2020a.

Bai, Y., Xu, D., Gu, K., Wu, X., Marinovic, A., Ro, C., Sun, Y., and Wang, W. Neural maximum common subgraph detection with guided subgraph extraction, 2020b. URL https://openreview.net/forum?id=BJgcwh4FwS.

Balcilar, M., Renton, G., Héroux, P., Gaüzère, B., Adam, S., and Honeine, P. Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*, 2021. URL https://openreview.net/forum?id=-qh0M9XWxnv.

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

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *ICLR*, 2017.

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

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

Cao, N., Yang, Z., Wang, C., Ren, K., and Lou, W. 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.

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

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *NeurIPS*, 2017.

Dai, H., Li, Y., Wang, C., Singh, R., Huang, P.-S., and Kohli, P. Learning transferable graph exploration. *NeurIPS*, 2019.

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

Fan, C., Zeng, L., Sun, Y., and Liu, Y.-Y. Finding key players in complex networks through deep reinforcement learning. *Nature Machine Intelligence*, 2(6):317–324, 2020.

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

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

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

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

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

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

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects. *ICML*, 2019.

Ling, X., Wu, L., Wang, S., Ma, T., Xu, F., Wu, C., and Ji, S. Hierarchical graph matching networks for deep graph similarity learning, 2020. URL https://openreview.net/forum?id=rkeqn1rtDH.

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

Liu, Y.-l., Li, C.-m., Jiang, H., and He, K. A learning based branch and bound for maximum common subgraph problems. *IJCAI*, 2019.

McCreesh, C., Ndiaye, S. N., Prosser, P., and Solnon, C. 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.

McCreesh, C., Prosser, P., and Trimble, J. A partitioning algorithm for maximum common subgraph problems. 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *NeurIPS Deep Learning Workshop 2013*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

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

Peyré, G., Cuturi, M., and Solomon, J. Gromov-wasserstein averaging of kernel and distance matrices. In *ICML*, pp. 2664–2672, 2016.

Solozabal, R., Ceberio, J., and Takáč, M. Constrained combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:2006.11984*, 2020.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *ICLR*, 2018.

Vismara, P. and Valery, B. 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.

Wang, R., Yan, J., and Yang, X. Learning combinatorial embedding networks for deep graph matching. *ICCV*, 2019.

Wang, R., Zhang, T., Yu, T., Yan, J., and Yang, X. Combinatorial learning of graph edit distance via dynamic embedding. *arXiv preprint arXiv:2011.15039*, 2020.

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

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

Xu, H., Luo, D., Zha, H., and Carin, L. Gromov-wasserstein learning for graph matching and node embedding. *ICML*, 2019b.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *ICLR*, 2019c.

Yan, X., Yu, P. S., and Han, J. Substructure similarity search in graph databases. In *SIGMOD*, pp. 766–777. ACM, 2005.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *ICML*, pp. 7134–7143. PMLR, 2019.

Yu, T., Wang, R., Yan, J., and Li, B. Learning deep graph matching with channel-independent embedding and hungarian attention. In *ICLR*, 2020. URL https://openreview.net/forum?id=rJgBd2NYPH.

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