GRAPH MATCHING BY GRAPH NEURAL NETWORK

BY

XIYANG LIU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Associate Professor Sewoong Oh

ABSTRACT

Graph matching or network alignment refers to the problem of matching two correlated graphs. This thesis presents a deep Q learning based method, which represents the matching process by a graph neural network. By breaking the symmetry, the parameterized graph neural network is able to capture a wide range of neighborhoods. Extensive experiments on various training and testing data have shown better performance, strong scalability and the ability to adapt to different domains.

To my mom Yuqiong Yang and dad Xinrong Liu, for their unconditional love and support.

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Sewoong Oh, for introducing this interesting problem to me and guiding me through the completion of this thesis with great patience. I also thank Kiran for fruitful discussions on the graph neural network and valuable suggestions.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

GNN	Graph Neural Network
PGM	Percolation Graph Matching

CHAPTER 1

INTRODUCTION

1.1 Problem Statement and Background

Graph structured data is becoming increasingly prevalent for modeling complex relationship among entities. Many types of real-world data, for example, the user data on social networks, gene data on biological regulatory networks, log data on telecommunication networks, or text documents on word embeddings all have a graph structure. Recently, both combinatorial methods and statistical models applicable on such network data have been active areas of research.

In this thesis, we address the graph matching/network alignment problem, where two partial views of the network are provided, but node identities are ambiguous. This problem has been widely studied in many scientific applications, such as computer vision, biology, social network analysis, and linguistics. Specifically, applications of graph matching include the following: (1) finding the same user in two anonymized social networks by graph structure [1], [2], [3]; (2) matching similar patterns in multiple images [4], [5], [6], [7], [8], [9]; (3) matching protein-protein interaction (PPI) networks in biology to understand the similarity between proteins of two species [10], [11], [12], [13]; (4) reconciling databases by aligning their database schema [14], [15]; and (5) other interesting applications on social network, chemistry or biology [16], [17].

In all the above applications, we often assume the two graphs to be correlated. For example, in the PPI network, as evolution transfers metabolic processes from species to species, PPI networks of two species can be seen as highly correlated. Then researchers can transfer their knowledge of proteins across species by identifying the correspondence of the protein in multiple PPI networks.

Graph matching/network alignment specifically refers to the problem of finding a bijective mapping of vertices from one graph to another. A mismatch for such mapping happens when two vertices are connected in one graph, while their mapped vertices are not connected in another. If two graphs are isomorphic, there exists a matching that has no mismatches. Graph matching is then reduced to graph isomorphism and this problem is called *exact graph matching*. However, in general, such perfect matching may not be feasible. In this case, graph matching tries to find a mapping that minimize the number of mismatches and this type of problem is said to be *inexact graph matching*.

As an example, a node represents a user in a social network and an edge is a unidirectional contact relationship with another user. We are given two graphs without node identities. $G_1(V_1, E_1)$ is the Facebook network of a group of users and $G_2(V_2, E_2)$ is the Twitter network of the same group. In this case, $|V_1| = |V_2| = n$. But the two graphs may not be isomorphic because some users may not add all their Facebook contacts on Twitter. Our task is to find one-to-one mapping $f : V_1 \to V_2$ such that any two users $(u_1, v_1) \in E_1$ iff $(f(u_1), f(v_1)) \in E_2$. This problem is more difficult when only the topologies of two graphs are available, i.e., the nodes are unlabeled. In this thesis, we mainly focus on the scenario that (1) two graphs have the same size, and (2) every node has to be matched to exactly one node in another graph.

Finding such optimal matching is computationally challenging. Let G_1 and G_2 be the adjacency matrices of two graphs. Mathematically, the problem above is expressed as finding a permutation matrix P such that:

$$P = \underset{X \in \Pi}{\operatorname{argmin}} ||G_1 - XG_2X^T||_F^2 \tag{1.1}$$

where Π is the set of all permutation matrices of size $n \times n$. By expanding the square in (1.1), we can obtain an equivalent quadratic assignment problem (QAP) formulation:

$$P = \underset{X \in \Pi}{\operatorname{argmax}} \operatorname{Tr}(G_1XG_2X^T) \tag{1.2}$$

QAP is NP-hard to solve exactly. It was also proven that QAP does not have an approximation algorithm running in polynomial time for any (constant) factor, unless P = NP [18]. As a special case of QAP, graph isomorphism is

to solve $||G_1 - XG_2X^T||_F^2 = 0$. It is still unclear if graph isomorphism is in P or not.

Recent advances in deep learning and specifically the graph neural networks have brought new insights into using deep neural nets to solve combinatorial problems on graphs approximately.

In this thesis, we present a new graph neural network based algorithm to solve graph matching problems. The empirical results show that our algorithm has better performance in terms of accuracy, scalability and speed compared to the state-of-the-art algorithms.

1.2 Related Works

Prior works on graph matching can be divided into the following categories:

Optimization: These works are based on relaxation of QAP problem (1.2). Zhao et al. [19] use the semidefinite programming (SDP) relaxation. While solving an SDP becomes unpractical when the graph size $n > 20$ due to the computational complexity. IsoRank [10] relaxed QAP to the integer quadratic program with some more relaxed constraints. Klau [20] alternatively uses a linear programming relaxation and solves it iteratively. Netalign [21] relaxed this problem as an integer quadratic programming problem and solves it with message passing.

Spectral Methods: EigenAlign and LowRankAlign proposed by Feizi et al. [22] are two spectral methods based on another relaxation of the QAP, which achieves high accuracy and time complexity of $O(n^2)$. But both SDP and spectral methods fail on regular graphs whose spectrums have repeated eigenvalues. There are other algorithms that use heuristics to approximate the solution.

Heuristic Methods: In the area of computational biology, many heuristic methods have been proposed to match the PPI networks, which typically minimize a convex cost function as a combination of structure similarity. These examples include: GRAAL [13], [12], MAGNA [11], and SPINAL [23].

Embeddings Methods: In the area of data mining, many graph matching problems are based on calculating the embedding of nodes and match them by similarity of embeddings. Generally, the embeddings can be computed by (1) local neighborhood [24], [25] and (2) recursive features [26],

which gather local neighborhood information iteratively and thus include information from a wider region. The embeddings are often gathered by heuristics and designed for a specific data domain.

Percolation Methods: Percolation graph matching (PGM) is one of the most scalable algorithms which is based on percolation theory. While it requires an initial seed to start the percolation. The performance of matching highly relies on the quality of the initial seed. It assigns the embeddings of each node based the matched subgraph. This line of research is widely analyzed under random graphs and implemented for social network deanonymization. PGM was first proposed independently by [1] and [27]. Improvement of the dependence of the initial seed was proposed by [28]. An analysis of a simple method of invoking the initial seed in a seedless setting was proposed in [29].

Graph Neural Network Based Embeddings: Nowak et al. [30] compute the embeddings of the nodes by a parameterized graph neural network, which resembles the recursive embedding in [26]. They trained the neural network in a supervised settings which assumes a ground truth. The accuracy of matched nodes is the best among all the existing work.

With the advancement of the graph neural network, another line of research focuses on applying the graph neural network on combinatorial problems.

Graph Neural Network: The graph neural network parameterizes the embedding of the nodes by applying linear combinations of local graph operators like graph adjacency or the graph Laplacian, and then applying nonlinear functions. It has better representational power of structured data and demonstrated breakthrough performance on tasks like link prediction, and graph classification [31]. Further improvement of the performance by attention mechanism was presented in [32] and [33].

Reinforcement Learning on Combinatorial Problems: Bello et al. [34] first proposed the applications of the deep reinforcement learning on combinatorial problems on graphs. In [35] and [36], the graph neural network was applied on several graph combinatorial problems. Kool and Welling [37] applied the attention network on the Euclidean TSP problem.

Among all the graph matching techniques, spectral methods proposed by Feizi et al. [22] and Nowak et al. [30] have the highest accuracies, which will be used as baselines in this thesis.

CHAPTER 2

GRAPH MATCHING BY PERCOLATION AND REINFORCEMENT LEARNING

2.1 Problem Definition

Given two undirected graph $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, $|V_1| = |V_2| = n$, our task is to find one-to-one mapping $f : V_1 \to V_2$ such that $(u_1, v_1) \in E_1$ iff $(f(u_1), f(v_1)) \in E_2$. We use $A_{1,2}$ to denote the adjacency matrix respectively. Then the problem is:

$$P = \underset{X \in \Pi}{\operatorname{argmin}} ||A_1 - X A_2 X^T||_F^2 \tag{2.1}$$

where Π is the set of all permutation matrices of size $n \times n$.

Let a *pair* (i, j) represent a pair of nodes in each graph, where $i, j \in V_{1,2}$. Let a *couple* $[u, v] \in V_1 \times V_2$ represent a pair of nodes across two graphs. We propose to solve this problem in a greedy manner, which matches one *couple* at each step until all the nodes are matched. The next couple we choose depends on the current matching. Let M be an $n \times n$ binary matrix where $M(u, v) = 1$ means that node u in network G_1 is mapped to node v in network G_2. In this setting, each node in one graph can be mapped to exactly one node in the other graph, i.e., $\sum_u M(u, v) = 1$ for all v, and similarly $\sum_v M(u, v) = 1$ for all u.

2.2 Percolation Graph Matching

Based on the percolation theory, percolation graph matching (PGM) requires an initial *seed*, which is a set of pre-matched couples $A_0 \subset V_1 \times V_2$. The goal is to find a bijective map, i.e. a set of couples $M \subset V_1 \times V_2$ such that every node appears in exactly **one** couple in M. PGM is a class of algorithms that iteratively expand A_0 to M by adding new couples $[i, j']$ at each step

to current *partial matching* A. This algorithm stops until all the possible couples have been added.

Input: $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, initial seed A_0 and threshold r
Output: The set of matched couples M
for *all couples* $[i, j'] \in A_0$ **do**
  add one score to all the neigboring couples of $[i, j']$
  if *the score of* $[r, s'] > r$ *and* $V_1(r) \notin E_1$ *and* $V_2(s') \notin E_2$ **then**
    add $[r, s']$ to A_0
  end
end
$M \leftarrow A_0$
return M

Algorithm 1: Percolation Graph Matching (PGM)

This algorithm is greedy as it picks the best couple to add to current matching at each iteration by looking at current partial matching. Alternatively, deep Q learning picks the best couple to add to current matching by looking at Q value which is global. So the idea of this thesis is to seek a way of learning a good Q function.

2.3 Reinforcement Learning Formulations

We aim to learn a parameterized Q function that measures the quality of long-term reward for the current action in the context of current matching. For a current matching S, the next couple $[u, v]$ to add to current S is chosen by maximizing Q. Let $t = \{0, 1, \cdots, n\}$ denote the number of steps. Specifically,

1. *State*: A state S is current matching. Initially, $S_0 = \emptyset$. It can be represented by an ordered list $S_t = ([u_1, v_1], [u_2, v_2], \cdots, [u_t, v_t])$.

2. *Action*: An action a is the current couple $a_t = [u_t, v_t]$ we pick from all the valid couples to add to the current matching.

3. *Transition*: The transition is deterministic here. $S_{t+1} = S_t \bigcup a_t$.

4. *Reward*: $r(S_t, a_t)$ at state S_t is defined as the change in total reward function $R = \sum_{i=1}^{n} r(S_t, a_t)$, which should be consistent with the matching loss.

5. *Policy*: During testing, we pick an action at each step by greedy policy $\pi(a_t|S_t) := \text{argmax}_a Q(S_t, a; G_1, G_2, \theta)$.

Figure 2.1 shows an example of the matching process. The blue unmatched candidate pair is chosen by considering the current matched pairs and unmatched nodes.

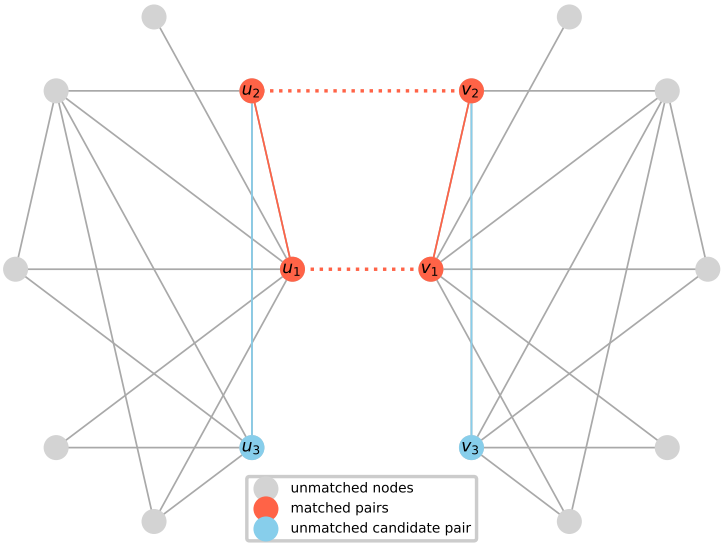

Figure 2.1: Illustration of state and action

2.4 Reward Function

The matching loss we consider for a permutation matrix is $X \in \Pi$ is $||A_1 - XA_2X^T||_F^2$. We propose a consistent reward function, which can be computed directly at each step via counting patterns in subgraphs.

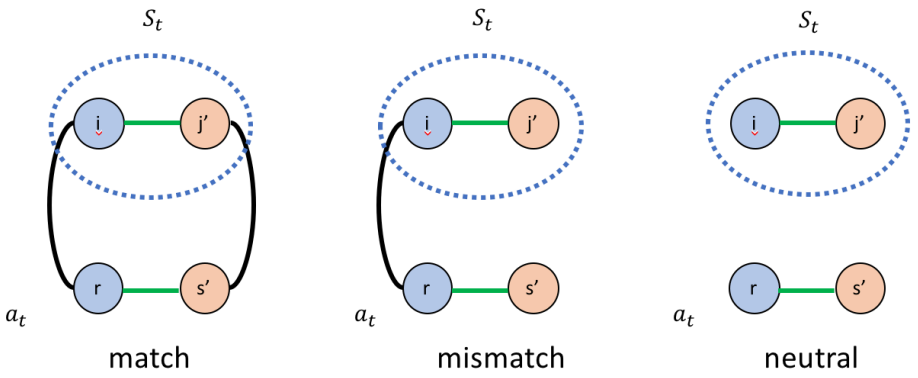

Figure 2.2: Illustration of match, mismatch, and neutral subgraphs

Definition 1. *Suppose* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *are undirected graphs. Let* $(i, r) \in V_1$ *and* $(j, s) \in V_2$ *where* $M(i, j) = 1$ *and* $M(r, s) = 1$. *Then,*

- (i, j) *and* (r, s) *are matches if* $(i, r) \in E_1$ *and* $(j, s) \in E_2$.
- (i, j) *and* (r, s) *are mismatches if only one of the edges* (i, r) *and* (j, s) *exists.*
- (i, j) *and* (r, s) *are neutrals if none of the edges* (i, r) *and* (j, s) *exists.*

Here we use the following reward function:

$$r(S_t, a_t) = \alpha_1 \times \#\text{ of matches} + \alpha_2 \times \#\text{ of mismatches} + \alpha_1 \times \#\text{ of neutrals} \tag{2.2}$$

2.4.1 Node Embeddings

We expect the Q function should take into account the current matching and also the structure similarity of two graphs. We first use a graph neural network to compute the embedding of each node, and then aggregate the embeddings to estimate Q values at the current step. We initialize the embedding $\mu_v^{(0)} \in \mathbb{R}^p$ at each node as zero vector. And for all $v \in V_1 \bigcup V_2$, we

update the embeddings at each iteration as:

$$\mu_v^{(t+1)} = \mathrm{BN}^{(t)}\mathrm{ReLU}(x_v\theta_1^{(t)} + (\sum_{u\in\Gamma(G_1)} \mu_u^{(t)})\theta_2^{(t)} + (\sum_{u\in\Gamma(G_2)} \mu_u^{(t)})\theta_2^{(t)} + (\sum_{u\in\Gamma(\tilde{G})} \mu_u^{(t)})\theta_3^{(t)}) \tag{2.3}$$

where $\theta_1^{(t)} \in \mathbb{R}^p$, $\theta_2^{(t)} \in \mathbb{R}^{p\times p}$, $\theta_3^{(t)} \in \mathbb{R}^{p\times p}$ are trainable parameters; $x_v = 1$ if v is already matched; $t = 0, 1, \cdots, T$ denotes the number of layers.

Alternatively, we can write this in a matrix form:

$$Y^{(t+1)} = \mathrm{BN}^{(t)}\mathrm{ReLU}(X\theta_1^{(t)} + \sum_J \begin{bmatrix} A_1^{(J)} & \\ & A_2^{(J)} \end{bmatrix} Y^{(t)}\theta_2^{(t)} + \begin{bmatrix} & M \\ M^\top & \end{bmatrix} Y^{(t)}\theta_3^{(t)} + b^{(t)}) \tag{2.4}$$

where $Y^{(t)} \in \mathbb{R}^{(n_1+n_2)\times p}$ is the concatenation of node embeddings of two graphs. $A_{1,2}^{(J)} = \text{zero diagnal}\min(1, A^J)$ encode J-hop neighborhoods of each node in each graph. This allows us to aggregate local information in different scales, which is useful to break the symmetry in regular graphs.

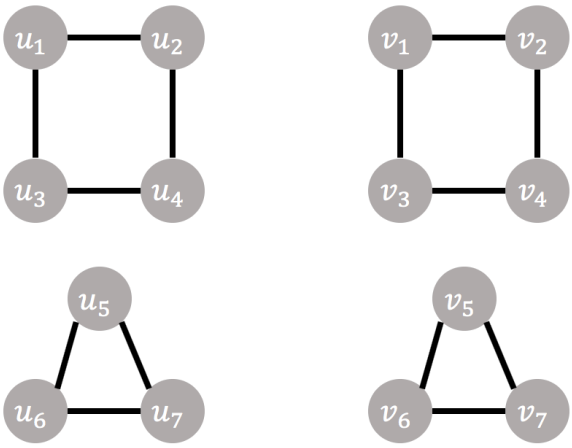

Figure 2.3: Example of breaking symmetry in regular graphs

Figure 2.3 gives us an example of the importance of wide range information. At first step, if we only encode one-hop neighborhoods, all the nodes have the same embedding, thus, there is a possibility to match u_1 to v_5. However,

two-hop neighborhoods allow us to break the symmetry between $u_{1,2,3,4}$ and $u_{5,6,7}$. It is natural to match two rectangles and two triangles in any order.

2.4.2 Parameterizing Q functions

We use the embedding at the last layer $Y^{(T)}$ to represent the Q function. For current matching S and action $[u, v]$, the Q value is computed by:

$$Q(S, [u, v]; G_1, G_2) = \\ \theta_4^T \mathrm{ReLU}(\mathrm{ReLU}([\sum_{u \in V_1} \mu_u^{(T)} \theta_5, \sum_{v \in V_2} \mu_v^{(T)} \theta_5, \mu_u^{(T)} \theta_6, \mu_v^{(T)} \theta_6, w(u, v)\theta_7])\theta_8 + b) \tag{2.5}$$

where $\theta_4 \in \mathbb{R}^{5p}\theta_5 \in \mathbb{R}^{p \times p}$,$\theta_6 \in \mathbb{R}^{p \times p}$, $\theta_7 \in \mathbb{R}^{6 \times p}$,$\theta_8 \in \mathbb{R}^{5p \times 5p}$,$b \in \mathbb{R}^{5p}$, $[\cdot, \cdot]$ is the concatenation operator.

$W = (A_1 M A_2, A_1 \mathbb{I}, \mathbb{I} A_2, A_1^2 M A_2^2, A_1^2 M A_2, A_1 M A_2^2) \in \mathbb{R}^{6 \times n \times n}$ is a tensor to break the symmetry for $[u, v]$ couple. Here $w(u, v) \in \mathbb{R}^6$ is six-dimensional vector.

As shown in Figure 2.4, after the first step, u_2, v_2, u_3, v_3 are matched. When we consider the next step, according to the symmetry of the current partially matched graph, the embedding of u_1, v_1, u_4, v_4 should be exactly the same. However, for possible remaining couples, if the Q value is simply represented by the embeddings, i.e. $Q = h(\mu_{u_{1,4}}, \mu_{v_{1,4}})$, it cannot distinguish between couple $[u_1, v_1]$ and $[u_1, v_4]$. Thus, we have to break the symmetry according to the current matched couples.

It is natural to count the number of paths through matched couples between any unmatched couple $[u, v]$ in Figure 2.5 to break the symmetry. The number of the first path between u and v is $[A_1 M A_2]_{u,v}$. Similarly, we can count the number of the other three paths. As we can see from Figure 2.4, although the embedding of u_1, v_1, u_4, v_4 are the same, the number of first paths between u_1 and v_1 is different from the number of second paths.

As shown in Figure 2.6, for any unmatched couple $[u, v]$ we use a six-dimensional vector W_{uv} to represent the counts of different paths. Together with the embeddings, we can parameterize the Q function.

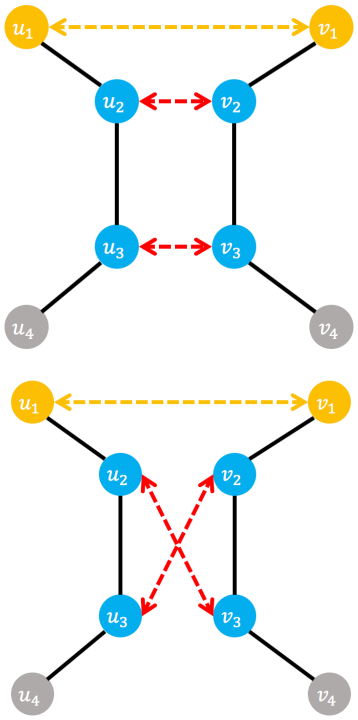

Figure 2.4: Example of breaking symmetry in matched graph

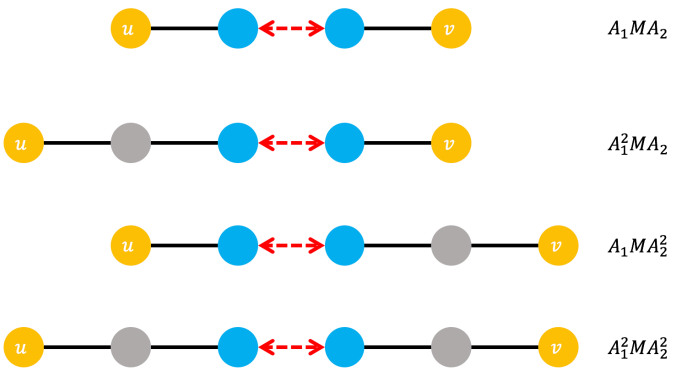

Figure 2.5: Different kinds of paths

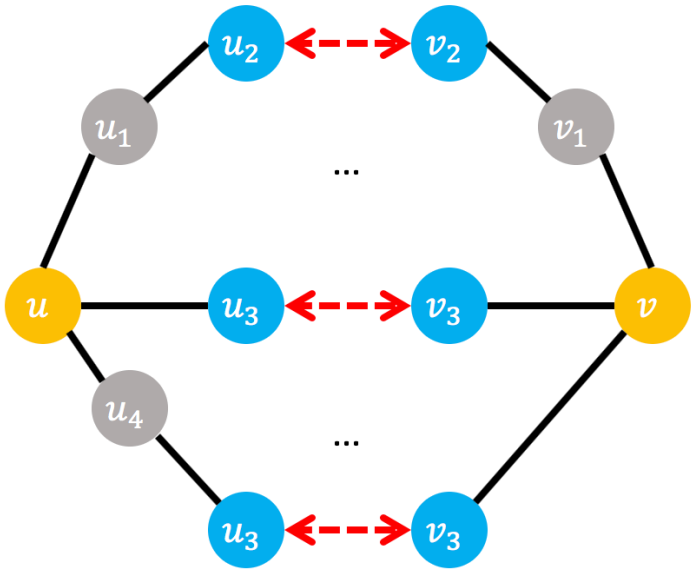

Figure 2.6: Counting paths between u and v

The overall neural network architecture is illustrated by Figure 2.7.

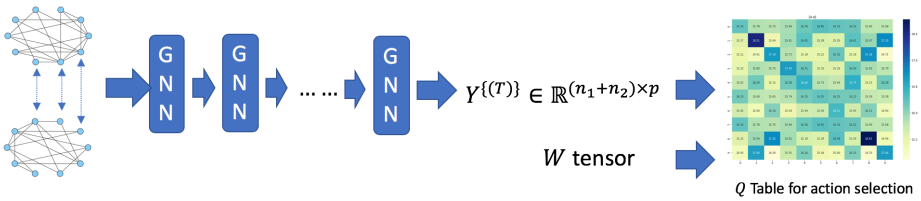

Figure 2.7: Neural network architecture

We first compute the embeddings of two graphs by a multilayer graph neural network to get Y. Then, we select the action by finding the largest valid unmatched couple in the Q table. This method can be seen as an extension of the PGM algorithm because they both use the number of matches added to partial matching as the measurement of quality of a new couple. However, DQN can find the first match by using only two graph structures without any initial seed. The graph neural network representation of the Q function allows us to encode the information we need (e.g. the number of neighboring couples), which provides more flexibility.

CHAPTER 3

TRAINING: Q LEARNING

3.1 Training Algorithm

We use the standard DQN training strategy [38], [39] for training the Q network.

Output: Q^*

Initialize replay memory D to capacity N

Initialize Q functions Q_1 and Q_2 with same weights $\theta_1 = \theta_2 = \theta$ **for** *episode* $e = 1, 2, ...E$ **do**

Draw random graphs (G_1, G_2) from some distribution

for *step* $t = 0, 1, 2, ...$ **do**

$$a_t = \begin{cases} \max_v Q_1(s_t, v), & w.r.p. \quad 1 - \epsilon \\ random \quad action, & w.r.p. \quad \epsilon \end{cases}$$

from environment, get r_t, s_{t+1}, add transition tuple (s_t, a_t, r_t, s_{t+1}) to D.

Sample a mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D.

set $y_j = \begin{cases} r_j & \text{terminate at} j+1 \\ r_j + \alpha \max_v Q_2(s_{j+1}, v) & \text{otherwise} \end{cases}$

perform a gradient descent step on square loss $(y_j - Q_1(s_j, a_j))$ w.r.t weights θ_1

every C steps, reset $Q_2 \leftarrow Q_1$ by copying weights

end

end

Algorithm 2: Training Algorithm

3.2 Validation and Testing

During testing, for each particular graph pair, we use a greedy policy to choose the next action. However, the Q function is computed by the matched nodes and structure of two graphs. When there are only a few matched couples, the Q function may not be accurate enough for picking the correct action. We apply the Q function guided tree search to alleviate the inaccurate Q function. As Figure 3.1 shows, instead of exhaustively searching for all the possible actions, at the t step, the agent only searches among the top B actions a that maximize the $Q(s_t, a)$. For each step, we search down D steps and pick the action that maximizes Q values or the sum of rewards after D steps. Thus, we can fix the breadth B and depth D of the tree. However, the tree search and back-tracking are still time-consuming. We set two thresholds to trigger the tree search. At the t step, we only do a tree search when t is less than T_0 and the gap of the top two Q values is less than G.

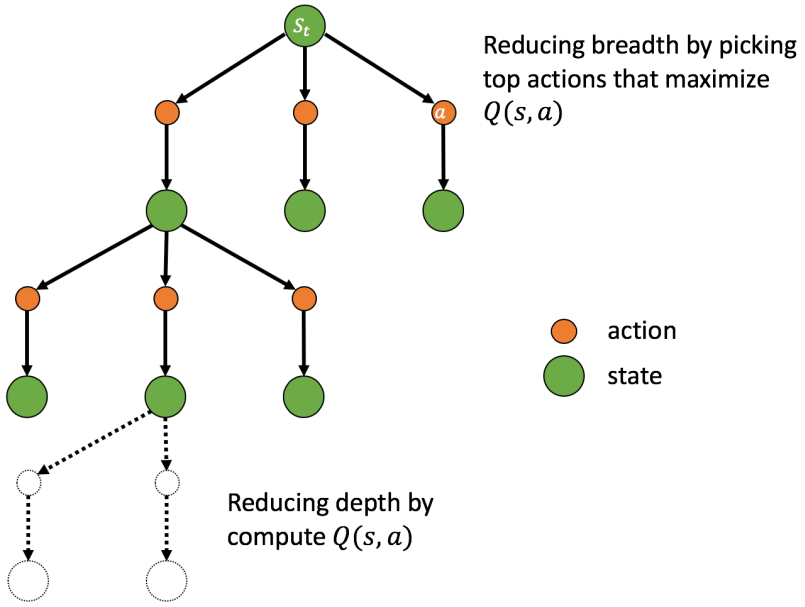

Figure 3.1: Q value guided tree search

Throughout the thesis, we set $B = 10$, $D = n$, $T_0 = 40$, $G = 3$ for testing all datasets.

CHAPTER 4

EXPERIMENT

4.1 Experiments Details

We demonstrate our methods on synthetic datasets.

1. Matching the Erdos-Renyi Graph: We consider A_1 to be an Erdos-Renyi graph with density p, A_2 is a small perturbation of A_1 according to the following model:

$$A_2 = A_1 \odot (1 - Q) + (1 - A_1) \odot Q' \tag{4.1}$$

   where Q and Q' are binary random matrices whose entries are drawn from i.i.d. Bernoulli distributions are such that $P(Q_{ij} = 1) = q$ and $P(Q'_{ij} = 1) = \frac{pq}{1-p}$.

2. Matching Random Regular Graph: We generate random regular graph A_1 by [40], A_2 is generated by the same small noise perturbation model according to (4.1).

3. Architecture: For our experiments, the number of layers is 10, the number of features is 32, and the batch size is 32.

4. Training: We use a learning rate of 10^{-4} with the Adam optimizer and initialize all of our parameters with the Gaussian distribution $\mathcal{N}(0, 10^{-6})$. We also use an exploration probability ϵ that is reduced from 1.0 to 0.05 linearly. The discount factor is set at 0.8.

4.2 Performance Comparisons

We evaluate our model by two measures: loss and recovery rate. For a matching, we compute the loss by using corresponded permutation matrix X as $L(X; A_1, A_2) = ||A_1 - XA_2X^T||_F^2$. According to the small noise perturbation model (4.1), we set the ground truth permutation matrix of both Erdos-Renyi graphs and random regular graphs as identity matrices. The recovery rate is simply measured by the average number of correct pairs compared with the ground truth. We compare the performance of LowRankAlign (k = 4), GNN and our RL model on the Erdos-Renyi graphs and random regular graphs. All graphs have the same size $n = 50$, same density $p = 0.2$, and the noise level q ranges from 0.00 to 0.05. For each noise levels, we report the mean and standard error on 100 experiments.

Our model is trained on $n = 50$, $p = 0.2$, $q = 0.05$ Erdos-Renyi graphs and tested on both Erods-Renyi graphs and random regular graphs.

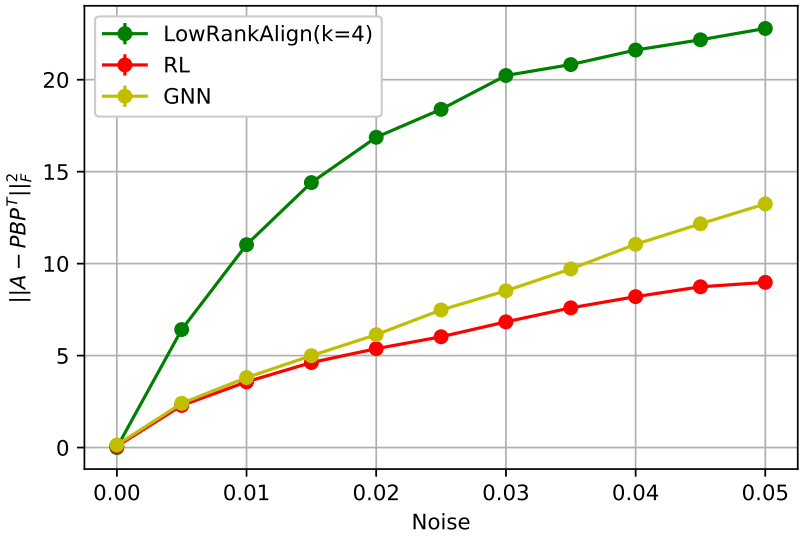

Figure 4.1: Loss on Erdos-Renyi graph

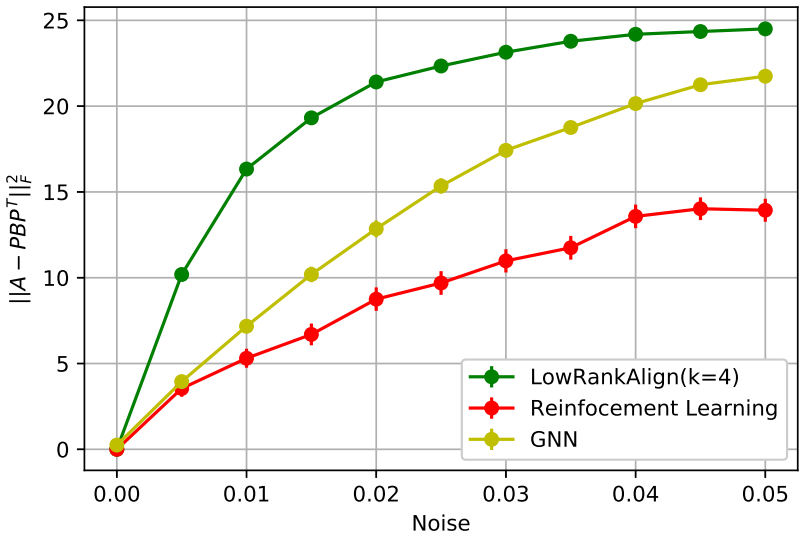

Figure 4.2: Loss on the regular graph

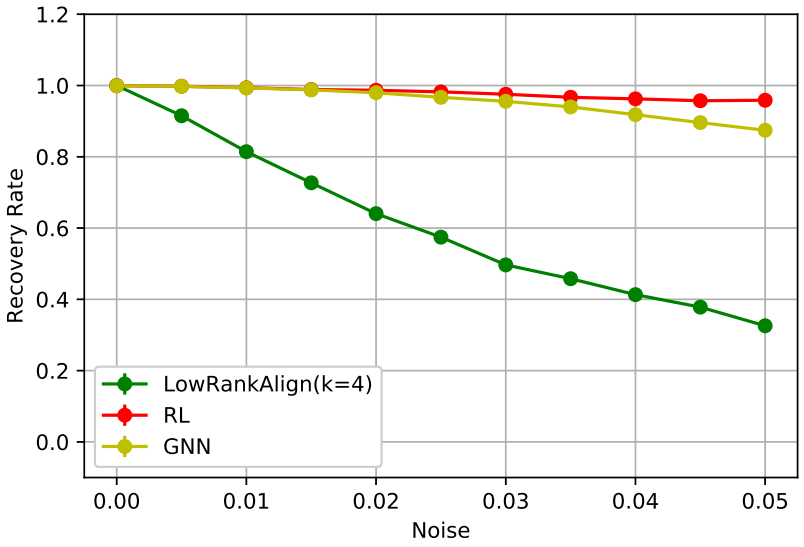

Figure 4.3: Recovery rate on Erdos-Renyi graph

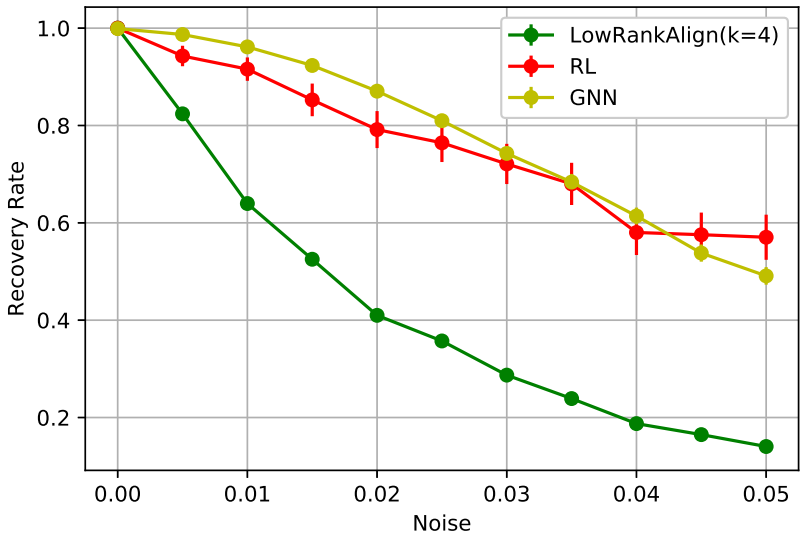

Figure 4.4: Recovery rate on the regular graph

We present our performance in Figures 4.1, 4.2, 4.3 and 4.4 by the metrics and standard error. We observe significant improvement of performance on the Erdos-Renyi graphs. Though our model and GNN have similar recovery rates on regular graphs, our model has lower losses.

We also compare the results on testing Erdos-Renyi graphs ($n = 50$, $p = 0.2$, $q = 0.03$); each experiment is repeated for 50 instances except the SDP method, which is repeated for five instances. The results are shown in Table 4.1.

Table 4.1: Performance comparisons of different algorithms

Methods	Loss	Recovery Rate
cSDP [41]	26.65 ± 0.49	$22.4\% \pm 3.86\%$
IsoRank [10]	25.25 ± 0.19	$35.93\% \pm 1.4\%$
Klau's [20]	26.58 ± 0.07	$11.3\% \pm 0.6\%$
NetAlign [21]	26.54 ± 0.07	$10.14\% \pm 0.5\%$
LowRankEigenAlign ($k = 8$,$c = 3$) [42]	24.23 ± 0.18	$43.1\% \pm 1.2\%$
LowRankAlign ($k = 4$) [22]	20.22 ± 0.16	$49.55\% \pm 0.9\%$
PGM ($a_0 = 10$) [27]	21.89 ± 0.11	$20.22\% \pm 1.7\%$
Canonical Labeling ($h = 10$) [29]	31.19 ± 0.07	$1.8\% \pm 0.3\%$
GNN [30]	8.52 ± 0.28	$95.5\% \pm 0.4\%$
RL (ours)	6.83 ± 0.33	$97.55\% \pm 0.6\%$

4.3 Generalization on Larger Graphs

To measure the scalability, we train our model on $n = 20$, $p = 0.2$, $q = 0.05$ Erdos-Renyi graphs and test on larger Erdos-Renyi graphs. As a comparison, the GNN model is trained on $n = 20$, $p = 0.2$, $q = 0.05$ Erdos-Renyi graphs with default hyper-parameters.

Table 4.2: Generalization on larger graphs and comparison with GNN

Train	RL($n = 20$)		GNN($n = 20$)	
	Loss	Recovery Rate	Loss	Recovery Rate
$n = 100$	15.89 ± 0.95	$95.64\% \pm 0.55\%$	44.68 ± 0.08	$25.44\% \pm 0.6\%$
$n = 200$	22.31 ± 2.41	$97.42\% \pm 0.82\%$	89.49 ± 0.06	$6.25\% \pm 0.17\%$
$n = 400$	77.73 ± 8.53	$87.09\% \pm 3.80\%$	178.94 ± 0.06	$1.63\% \pm 0.04\%$

As Table 4.2 shows, the rows are testing performance with various sizes, the columns are different training data. Our method has much higher generalization performance compared to the GNN method. It can match two graphs that have size $n = 400$ by training on small graphs ($n = 20$). We omit the results when $n > 400$ due to computational time restrictions.

4.4 Generalization across Graph Types

To understand what kind of training data has better generalization ability on other graphs, we train our model on small noise graphs, random regular graphs with fixed size $n = 50$ and density $p = 0.2$.

Table 4.3: Generalization across training data types

Noise q	$q = 0.01$ Erdos-Renyi		$q = 0.05$ Regular	
	Loss	Recovery Rate	Loss	Recovery Rate
0.01 ER	12.81 ± 0.93	$75.2\% \pm 4.3\%$	25.71 ± 0.15	$2.56\% \pm 0.9\%$
0.01	10.53 ± 1.14	64.92%6.4%	23.37 ± 0.06	$2.88\% \pm 0.5\%$
0.05 ER	15.67 ± 0.75	$63.6\% \pm 5.3\%$	25.50 ± 0.09	$1.2\% \pm 0.2\%$
0.05	15.89 ± 0.90	$45.4\% \pm 6.4\%$	23.76 ± 0.07	$2\% \pm 0.3\%$

Table 4.3 shows that training on Erdos-Renyi graphs has the most generalization power. Training on graphs with large noise can generalize on the

graphs with smaller noise, which verifies the intuition that training on harder examples can lead to generalization on easier samples.

4.5 Simulation Time

The testing time is directly related to the depth, width and threshold of the tree search. For all sizes of graphs, we use the settings for the tree search and report the average testing time among 50 instances (see Table 4.4). All of our experiments are computed on one single GPU.

Table 4.4: Simulation time

	Time(seconds)
$n = 20$	19
$n = 50$	69
$n = 100$	314
$n = 200$	872
$n = 400$	2469

CHAPTER 5

CONCLUSION

In this thesis, we presented a reinforcement learning-based algorithm for matching two correlated graphs. The main contribution of the algorithm is the architecture of the graph neural network, which breaks the symmetry of a matched subgraph. The reward design and shaping makes it flexible under other settings like overlapping matching or different sizes. We demonstrated better performance in terms of the number of mismatches on various datasets. By training on small but hard examples, our method is able to generalize to larger graphs and graphs from different distributions, which makes it useful in real datasets.

REFERENCES

[1] N. Korula and S. Lattanzi, "An efficient reconciliation algorithm for social networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 5, pp. 377–388, 2014.

[2] K. Sharad and G. Danezis, "An automated social graph de-anonymization technique," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014, pp. 47–58.

[3] K. Sharad, "Learning to de-anonymize social networks," Ph.D. dissertation, Computer Laboratory, University of Cambridge, 2016.

[4] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[5] T. Cour, P. Srinivasan, and J. Shi, "Balanced graph matching," in *Advances in Neural Information Processing Systems*, 2007, pp. 313–320.

[6] F. Zhou and F. De la Torre, "Factorized graph matching," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 127–134.

[7] F. Zhou and F. De la Torre, "Spatio-temporal matching for human pose estimation in video." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 8, pp. 1492–1504, 2016.

[8] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei, "Neural graph matching networks for fewshot 3d action recognition," in *European Conference on Computer Vision*. Springer, 2018, pp. 673–689.

[9] A. Zanfir and C. Sminchisescu, "Deep learning of graph matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2684–2693.

[10] R. Singh, J. Xu, and B. Berger, "Global alignment of multiple protein interaction networks with application to functional orthology detection," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105, no. 35, 2008, p. 12763.

[11] V. Saraph and T. Milenković, "MAGNA: Maximizing accuracy in global network alignment," *Bioinformatics*, vol. 30, no. 20, pp. 2931–2940, 2014.

[12] N. Malod-Dognin and N. Pržulj, "L-GRAAL: Lagrangian graphlet-based network aligner," *Bioinformatics*, vol. 31, no. 13, pp. 2182–2189, 2015.

[13] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj, "Topological network alignment uncovers biological function and phylogeny," *Journal of the Royal Society Interface*, p. rsif20100063, 2010.

[14] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 963–972.

[15] S. Zhang, J. Yang, and W. Jin, "SAPPER: Subgraph indexing and approximate matching in large graphs," in *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[16] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively multitask networks for drug discovery," *arXiv preprint arXiv:1502.02072*, 2015.

[17] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[18] S. Sahni and T. Gonzalez, "P-complete approximation problems," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 555–565, 1976.

[19] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz, "Semidefinite programming relaxations for the quadratic assignment problem," *Journal of Combinatorial Optimization*, vol. 2, no. 1, pp. 71–109, 1998.

[20] G. W. Klau, "A new graph-based method for pairwise global network alignment," *BMC Bioinformatics*, vol. 10, no. 1, p. S59, 2009.

[21] M. Bayati, D. F. Gleich, A. Saberi, and Y. Wang, "Message-passing algorithms for sparse network alignment," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 7, no. 1, p. 3, 2013.

[22] S. Feizi, G. Quon, M. Recamonde-Mendoza, M. Medard, M. Kellis, and A. Jadbabaie, "Spectral alignment of graphs," *arXiv preprint arXiv:1602.04181*, 2016.

[23] A. E. Aladağ and C. Erten, "SPINAL: Scalable protein interaction network alignment," *Bioinformatics*, vol. 29, no. 7, pp. 917–924, 2013.

[24] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *AAAI*, 2017, pp. 2429–2435.

[25] M. Heimann, H. Shen, T. Safavi, and D. Koutra, "Regal: Representation learning-based graph alignment," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 117–126.

[26] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos, "It's who you know: Graph mining using recursive structural features," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2011, pp. 663–671.

[27] L. Yartseva and M. Grossglauser, "On the performance of percolation graph matching," in *Proceedings of the First ACM Conference on Online Social Networks*. ACM, 2013, pp. 119–130.

[28] E. Kazemi, S. H. Hassani, and M. Grossglauser, "Growing a graph matching from a handful of seeds," in *Proceedings of the VLDB Endowment*, vol. 8, no. 10, 2015, pp. 1010–1021.

[29] O. E. Dai, D. Cullina, N. Kiyavash, and M. Grossglauser, "On the performance of a canonical labeling for matching correlated Erdos-Renyi graphs," *arXiv preprint arXiv:1804.09758*, 2018.

[30] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, "A note on learning algorithms for quadratic assignment with graph neural networks," *arXiv preprint arXiv:1706.07450*, 2017.

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[32] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, vol. 1, no. 2, 2017.

[33] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," *arXiv preprint arXiv:1803.03735*, 2018.

[34] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[35] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *arXiv preprint arXiv:1704.01665*, 2017.

[36] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Graph2seq: Scalable learning dynamics for graphs," *arXiv preprint arXiv:1802.04948*, 2018.

[37] W. Kool and M. Welling, "Attention solves your TSP," *arXiv preprint arXiv:1803.08475*, 2018.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[39] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[40] J. H. Kim and V. H. Vu, "Generating random regular graphs," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. ACM, 2003, pp. 213–222.

[41] J. F. B. Ferreira, Y. Khoo, and A. Singer, "Semidefinite programming approach for the quadratic assignment problem with a sparse graph," *Computational Optimization and Applications*, vol. 69, no. 3, pp. 677–712, 2018.

[42] H. Nassar, N. Veldt, S. Mohammadi, A. Grama, and D. F. Gleich, "Low rank spectral network alignment," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. International World Wide Web Conferences Steering Committee, 2018, pp. 619–628.