Deep Reinforcement Learning of Graph Matching

Chang Liu
Shanghai Jiao Tong University
Shanghai, China
only-changer@sjtu.edu.cn

Runzhong Wang
Shanghai Jiao Tong University
Shanghai, China
runzhong.wang@sjtu.edu.cn

Zetian Jiang
Shanghai Jiao Tong University
Shanghai, China
maple_jzt@sjtu.edu.cn

Junchi Yan
Shanghai Jiao Tong University
Shanghai, China
yanjunchi@sjtu.edu.cn

Abstract

Graph matching (GM) under node and pairwise constraints has been a building block in areas from combinatorial optimization, data mining to computer vision, for effective structural representation and association. We present a reinforcement learning solver for GM i.e. RGM that seeks the node correspondence between pairwise graphs, whereby the node embedding model on the association graph is learned to sequentially find the node-to-node matching. Our method differs from the previous deep graph matching model in the sense that they are focused on the front-end feature extraction and affinity function learning, while our method aims to learn the back-end decision making given the affinity objective function whether obtained by learning or not. Such an objective function maximization setting naturally fits with the reinforcement learning mechanism, of which the learning procedure is label-free. These features make it more suitable for practical usage. Extensive experimental results on both synthetic datasets, Willow Object dataset, Pascal VOC dataset, and QAPLIB showcase superior performance regarding both matching accuracy and efficiency. To our best knowledge, this is the first deep reinforcement learning solver for graph matching.

1. Introduction

Graph Matching (GM) aims to find node correspondence between pairwise graphs, which is fundamental in vision applications e.g. image keypoint matching [7], person re-identification [37], image retrieval [17], etc. In its general form, GM can be formulated as a combinatorial optimization problem namely Lawler's Quadratic Assignment Problem (Lawler's QAP) [23]. Generally speaking, solving a graph matching problem includes two steps: extracting features from input images to formulate the QAP instance and solving the QAP instance, namely **front-end** feature extractor and **back-end** solver, respectively.

The recent efforts in developing supervised learning based deep graph matching networks have shown the potential of deep learning for graph matching by adopting CNNs for visual features [51], graph neural networks for graph structures [41, 50], and geometric learning on structural features [13, 52]. In general, these works mainly focus on learning front-end feature extractors from real-world input data with ground truth labels as supervision. They simply combine their front-end feature extractors with some traditional combinatorial solvers for the QAP optimization, which means they hardly use deep learning technologies to improve the back-end solvers.

However, focusing only on the front-end part and ignoring the back-end will limit the development of deep graph matching research. In this sense, utilizing label-free reinforcement learning algorithms for designing a new back-end solver becomes a promising tool for pushing the frontier of graph matching research. To our surprise, we have not identified any prior work in this direction. Back-end solver aims to solve the QAP by maximizing the affinity score, which is a suitable platform for reinforcement learning that does not need any ground truth permutations as labels. In this paper, we propose the first **R**einforcement learning based **G**raph **M**atching back-end solver *RGM*. Though utilizing the power of deep reinforcement learning, *RGM* should be compared with other back-end solvers rather than existing deep GM methods that focus on front-end feature extraction. The comparison between *RGM* and traditional back-end solvers is relatively fair since *RGM* does not require ground truth labels and split the train/test set, which

means *RGM* uses the same input as traditional back-end solvers and has prevented over-fitting.

We propose three key components to entail the success of *RGM*. First, the agent is applied to the association graph as derived from input pairwise graphs, which allows the agent to focus on one graph and decouple the link between two individual graphs. Second, novel graph neural networks aid in building the networks that can better learn the representation and choose actions with more considerate embedding. Third, we adopt the Double Dueling DQN [30, 46, 15, 16] algorithm with priority experience replay memory [35].

In a nutshell, the highlights of this paper are:

1) We show how reinforcement learning solver, for the first time to our best knowledge, can be developed for graph matching, which is so far missing in the literature despite the active research in this area. The state representation and action space in *RGM* are based on the association graph, which allows for direct node embedding in a single homogeneous node space to enable network learning.

In essence, it is important to note that our approach is focused on learning the back-end QAP solver. In contrast, existing deep graph matching methods are learning the front-end feature extractor. In this spirit, our technique is orthogonal and supplementary to the prior works, and it opens a new direction for combing both as an open question.

2) The resulting *RGM* solver enjoys two functional merits over the existing deep learning based GM methods: i) the learning is free from the tedious ground truth permutation labeling which can even be impossible for large-scale datasets in practice; ii) the sequential decision making procedure naturally allows for its applicability in graph matching scenarios with initial seeds, and thus enhances its usability. In contrast, existing learning based methods all generate the matching in one-shot and lack the mechanism to leverage the given seed node correspondence information.

3) Across synthetic datasets, Willow Object dataset, Pascal VOC dataset, and QAPLIB benchmarks, our approach *RGM* consistently shows competitive performance in both accuracy and objective score, compared with non-learning baselines. It is important to note that *RGM* focuses on learning the back-end solver and hence it is orthogonal to existing deep GM methods that focus on boosting the front-end feature extraction, since they focus on different parts and should be complementary rather than competitive.

It is worth noting that state-of-the-art front-end deep GM models [41, 42, 44, 32] outperform *RGM* on the accuracy, which is not surprising in that they are all supervised models while ours is label-free as *RGM* only counts on the matching affinity signal which might not be reliable in noisy cases. For vision problems, learning effective graph representations including node features can be probably a more effective way for improving accuracy, especially consider-

ing our RL based back-end solver is an early learnable back-end solver that still in its infancy. Yet our method opens the space for back-end learning, for which existing literature is almost using non-learnable modules e.g. Sinkhorn nets [9]. Notably *RGM* has outperformed the seminal deep GM method GMN [51] as shown in our experiments. We leave for the integration of these two orthogonal techniques for GM to future work.

2. Related Work

We discuss the existing works in three areas as closely related to ours: i) graph matching as the problem we address in this paper; ii) deep learning of graph matching, which is the emerging line of research in graph matching; and iii) reinforcement learning for combinatorial optimization as our work readily falls into this more broad category.

Graph matching. It aims to find the node correspondence by considering both the node features as well as the edge attributes, which is known as NP-hard in its general form [49, 48, 27]. Moreover, graph matching falls into a more general so-called Quadratic Assignment Problem (QAP). Classic methods mainly resort to different optimization heuristics ranging from a random walk [8], spectral matching [24], path-following algorithm [53], graduated assignment [14], to SDP relaxation based technique [36], etc. However, such optimization based methods are in face of saturation of performance, and hence learning based approaches are receiving more and more attention recently.

Deep learning for graph matching. Deep neural networks have been an attractive paradigm for solving combinatorial problems [3] including graph matching [49]. Most existing works either use supervised learning or imitation learning to train the network, which is often enhanced with the Sinkhorn-network [9] as the key component to fulfill double-stochasticity for output matching estimation. There have been methods under this paradigm [41, 50, 44]. Meanwhile, in a recent work [33], the authors propose the black-box optimization pipeline to incorporate existing non-trainable graph matching solvers into end-to-end learning. Readers are referred to [48] for a more comprehensive overview of recent developments in this direction.

Reinforcement learning for combinatorial optimization. There is growing interest in using reinforcement learning in solving combinatorial optimization problems [3, 29]. Researchers have considered value based [30] and policy based [38] reinforcement learning in some NP-hard combinatorial optimization problems, such as traveling salesman problem (TSP) [20, 19, 5], vehicle routing problem (VRP) [31, 11, 2], job scheduling problem (JSP) [6, 28], bipartite matching [45], maximal common subgraph (MCS) [1], causal discovery [55]. The main challenges of these approaches are designing suitable problem representation and tuning reinforcement learning algorithms.

For combinatorial optimization problems on single graph, pointer networks [40] and graph neural networks [54] are the most widely used representations. However, for graph matching, there are two graphs for input and the agent needs to pick a node from each of the two graphs every step, which differs from these aforementioned single-graph combinatorial problems.

So far, there has been no (deep) RL approach for graph matching, which we think is an important direction for pushing the frontier of GM in learning the back-end solver.

3. Preliminaries

Graph matching aims to find nodes correspondence among two or multiple graphs, which is a fundamental problem in computer vision and pattern recognition. In this paper, we mainly focus on two graph matching, which is also known as pairwise graph matching.

Given two weighted graphs G^1 and G^2, we need to find the matching between their nodes such as their affinity score can be maximized. Here, we use V^1 and V^2 to represent the nodes of graph G^1 and G^2. For simplicity, we suppose that $|V^1| = n_1, |V^2| = n_2, n_1 \leq n_2$, which means all nodes in G^1 must be matched to G^2 but there can be outliers in G^2. E^1 and E^2 denote the edge attributes of graph G^1 and G^2. The affinities of pairwise graph matching include the first order (node) affinities and the second order (edge) affinities. Generally, the graph matching problem can be regarded as Lawler's Quadratic Assignment Problem [23]:

$$J(\mathbf{X}) = \text{vec}(\mathbf{X})^\top \mathbf{K} \text{vec}(\mathbf{X}),$$
$$\mathbf{X} \in \{0,1\}^{n_1 \times n_2}, \ \mathbf{X}\mathbf{1}_{n_2} = \mathbf{1}_{n_1}, \ \mathbf{X}^\top \mathbf{1}_{n_1} \leq \mathbf{1}_{n_2} \tag{1}$$

where $\mathbf{X}$ is the permutation matrix of which the element is 0 or 1, $\mathbf{X}_{i,a} = 1$ denotes node i in graph G^1 is matched with node a in graph G^2. $\text{vec}(\cdot)$ means column-vectorization. $\mathbf{K} \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$ is the affinity matrix. For node i in G^1 and node a in G^2 the node-to-node affinity is encoded by the diagonal element $\mathbf{K}_{ia,ia}$, while for edge ij in G^1 and edge ab in G^2 the edge-to-edge affinity is encoded by the off-diagonal element $\mathbf{K}_{ia,jb}$. Assuming i, a both start from 0, the index ia means $i \times n_2 + a$. The objective of Lawler's QAP is to maximize the sum of both first order and second order affinity score $J(\mathbf{X})$ given the affinity matrix $\mathbf{K}$ by finding an optimal permutation $\mathbf{X}$.

By definition, the graph matching problem contains two input graphs. Instead of directly working on two individual graphs, in this paper, we decide to first construct the association graph of the pairwise graphs as a new representation [24, 42], making the problem more easily and efficiently handled by reinforcement learning. We construct an association graph $G^a = (V^a, E^a)$ from the original pairwise graph G^1 and G^2, with the help of the affinity matrix $\mathbf{K}$. We merge each two nodes $(v_i, v_a) \in V^1 \times V^2$ as a

vertex $v_p \in V^a$. [1] We can see that the association graph contains $|V^a| = n_1 \times n_2$ vertices. There exists an edge for every two vertices as long as they do not contain the same node from the original pairwise graphs, so every vertex is connected to $(n_1 - 1) \times (n_2 - 1)$ edges. There exist both vertex weights $w(v_p)$ and edge weights $w(v_p, v_q)$ in the association graph. By definition, the vertex and edge weights denote the first and second order affinities of Lawler's QAP, respectively:

$$\mathbf{F}_{pp} = w(v_p) = \mathbf{K}_{ia,ia}, \text{ where } p = ia$$
$$\mathbf{W}_{pq} = w(v_p, v_q) = \mathbf{K}_{ia,jb}, \text{ where } p = ia, q = jb \tag{2}$$

where the vertex index p in the association graph G^a means a combination of the indices i and a in the original pairwise graphs G^1 and G^2. $\mathbf{F}, \mathbf{W} \in \mathbb{R}^{n_1 n_2}$ are the weight matrices that contain the vertex weights and edge weights in the association graph, respectively. Figure 1 shows an example of how we construct the association graph from the pairwise graphs. Selecting a vertex p in the association graph is equal to matching nodes i and a in the original pairwise graphs.

After constructing the association graph, we can rewrite the original objective function in Equation 1. In the association graph G^a, we select a set of vertices $\mathbb{U}$, which forms a subgraph of G^a and is also a complete graph. The set of vertices $\mathbb{U}$ in the association graph is equivalent to the permutation matrix $\mathbf{X}$, as long as the set $\mathbb{U}$ does not violate the constraint in Equation 1 (will be discussed later in Section 4.2). Besides, the original objective score in Equation 1 can be regarded as maximizing the sum of the vertex weights (original first order affinities) and edge weights (original second order affinities) in the complete graph formed by $\mathbb{U}$:

$$J(\mathbb{U}) = \sum_{v_p \in \mathbb{U}} w(v_p) + \sum_{v_p, v_q \in \mathbb{U}} w(v_p, v_q) \tag{3}$$

Then, we can optimize the new objective function in Equation 3 on the association graph by learning how to add suitable vertices to the set $\mathbb{U}$.

4. RL for Graph Matching: a back-end Solver

In this section, we propose our deep reinforcement learning based solver *RGM* for graph matching, in the sense of maximizing the affinity objective function. We will first introduce the model framework in Section 4.1. Then, we show the design of our reinforcement learning agent in Section 4.2. We further describe the network structure in Section 4.3. Finally, we describe the experience replay memory in Section 4.4 and the updating algorithm in Section 4.5.

---

[1]To distinguish between graphs to be matched and the association graph, in this paper, we use "node" for the original pairwise graphs and "vertex" for the association graph.
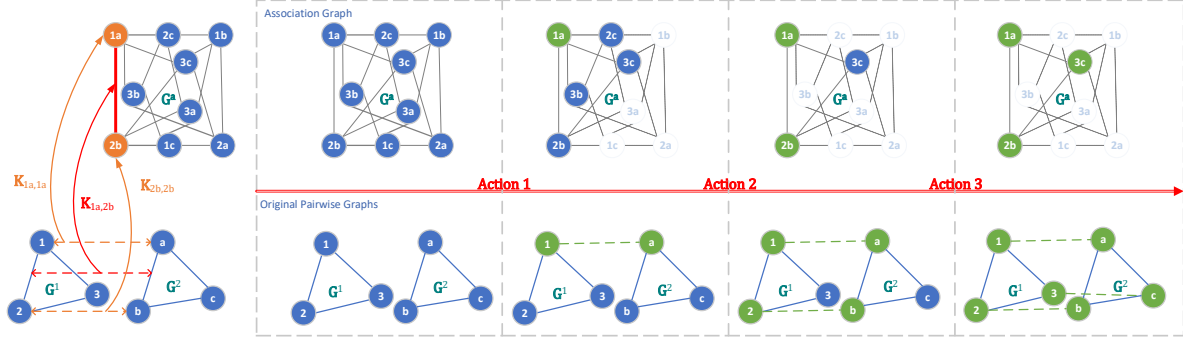
Figure 1. The relation between the pairwise graphs and the association graph, where G^1, G^2 on the bottom are the pairwise graphs and G^a on the top is the association graph. The construction of the association graph from the pairwise graphs is shown on the left. We show the matching process on the right: the blue vertices denote available vertices, the green vertices denote selected (or equivalently, matched) vertices, and the blurted vertices denote unavailable vertices. The agent selects "1a", "2b", and "3c" progressively on the association graph.

4.1. Framework

The fundamental idea of reinforcement learning is to learn from the interactions between the agent and the environment [39]. The agent's observation of the current environment is called state s. The agent chooses an action a given the current state by a specific policy. After the agent performs an action, the environment will transfer to another state s'. Meanwhile, the environment will feedback to the agent with a reward r. This pipeline solves the problem progressively. For graph matching, "progressively" means to select the vertex in the association graph one by one. The environment is defined as a partial solution to the original combinatorial problem (Equation 1) and equivalently the association graph, where the reward denotes the improvement of the objective function by matching a new pair of nodes. The interactions between the agent and the environment are recorded as a transition (s, a, r, s') into an experience replay memory $\mathcal{M}$. After several episodes, the agent updates its neural networks f_θ according to the transitions sampled from $\mathcal{M}$. We design our reinforcement learning method based on Double Dueling DQN [30, 46, 15, 16], where the training process is described in Algorithm 1.

4.2. Agent Design

First, we show the details of state, action, and reward:

1) State. Our state is the combination of the current partial solution $\mathbb{U}'$, the weight matrix $\mathbf{W}$ and the adjacent matrix $\mathbf{A}$ on the association graph. The partial solution $\mathbb{U}'$ is also a set of vertices in the association graph, with $|\mathbb{U}'| \leq n_1$. The size of $\mathbb{U}'$ raise from 0 at the beginning of each episode, and finally reaches n_1 in the end, which means partial solution $\mathbb{U}'$ becomes a complete solution $\mathbb{U}$.

2) Action. The action of our reinforcement learning agent is to select a vertex in the association graph and add it to the partial solution $\mathbb{U}'$ until n_1 vertices are selected. There is one constraint for selecting the next vertex by the

---

Algorithm 1: Training Process of *RGM*

Input: Dataset $\mathbb{D}$; step size η; exploration rate ϵ; updating frequency c_1, c_2

1 Randomly initialize Q-value network f_θ;
2 Initialize target Q-value network $f_{\theta^-} \leftarrow f_\theta$;
3 Initialize experience replay memory $\mathcal{M}$;
4 $cnt \leftarrow 0$;
5 **for** *episode* $\longleftarrow$ *0, 1, 2, …* **do**
6   Sample pairwise graphs G^1, G^2 from dataset $\mathbb{D}$;
7   Construct the association graph G^a;
8   Acquire the initialization state s in G^a;
9   **for** *ind* $\longleftarrow$ *0, 1, 2, …, n_1* **do**
10   Estimate Q-value given state s by f_θ;
11   With probability ϵ select a random action a otherwise select $a = \arg\max_a \mathbf{Q}(s, a; f_\theta)$;
12          $cnt \leftarrow cnt + 1$;
13   Interact with the association graph G^a $s', r \leftarrow s, a$;
14   Store the transition (s, a, r, s') in $\mathcal{M}$;
15   **if** $cnt \% c_1 == 0$ **then**
16   Calculate $\mathcal{L}(f_\theta; \mathcal{M})$ by Equation 9;
17   Update $f_\theta : \theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(f_\theta; \mathcal{M})$;
18   Update the transition priority in $\mathcal{M}$;
19   **if** $cnt \% c_2 == 0$ **then**
20   Update $f_{\theta^-} : f_{\theta^-} \leftarrow f_\theta$;
21          $s \leftarrow s'$

---

definition of graph matching that we can not match two nodes in G^1 to the same node in G^2 and vice versa. In Figure 1, once we select the vertex "1a", it means we have matched node "1" in G^1 and node "a" in G^2. Then, we can not match node "1" to node "b" or "c" later, which means we can not select vertices "1b" or "1c". Given the current

partial solution $\mathbb{U}'$, the available vertices set $\mathbb{V}$ is written as:

$$\mathbb{V} = \{v \mid \mathbf{A}(v, v') = 1 \ \forall \ v' \in \mathbb{U}', \ v \in V^a\} \qquad (4)$$

where V^a is the vertices in the association graph, $\mathbf{A}$ is the adjacent matrix in the association graph. Equation 4 is correct since two vertices are connected if they do not contain the same node from the original pairwise graphs. If a vertex is connected to all vertices in $\mathbb{U}'$, then it has no conflict.

Then, the action of our reinforcement learning agent is to pick a node v from the available vertices set $\mathbb{V}$ as Equation 5 shows, where $\mathbb{U}_{old}$ is the old partial solution, and $\mathbb{U}_{new}$ is the new partial solution after an action.

$$\mathbb{U}_{new} = \mathbb{U}_{old} \cup \{v\}, v \in \mathbb{V} \qquad (5)$$

3) Reward. We define the reward as the improvement of the objective score between the old partial solution and the new partial solution after executing an action. With the help of Equation 3, we can write our reward function as:

$$r = J(\mathbb{U}_{new}) - J(\mathbb{U}_{old}) \qquad (6)$$

4.3. Network Structure

4.3.1 State Representation Networks

To better represent the current state on the association graph, we choose graph neural networks (GNN) [21, 34] to compute its embedding. GNN extracts the vertex features based on their adjacent neighbors. However, traditional GNN is not sensitive to edge weights. To better use the edge weights in the association graph, we derive from the idea of struct2vec [10]. In our embedding networks, the current solution, node weights, and edge weights of the association graph are considered. The embedding formula is:

$$
\begin{aligned}
\mathbf{E}^{t+1} &= \text{ReLU}(\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3 + \mathbf{h}_4) \\
\mathbf{h}_1 &= \mathbf{X}' \cdot \theta_1^\top \\
\mathbf{h}_2 &= \frac{\mathbf{A} \cdot \mathbf{E}^t}{(n_1 - 1)(n_2 - 1)} \cdot \theta_2 \\
\mathbf{h}_3 &= \frac{\mathbf{A} \cdot \mathbf{F}}{(n_1 - 1)(n_2 - 1)} \cdot \theta_3^\top \\
\mathbf{h}_4 &= \frac{\sum(\text{ReLU}(\mathbf{W} \cdot \theta_5))}{(n_1 - 1)(n_2 - 1)} \cdot \theta_4
\end{aligned}
\qquad (7)
$$

where $\mathbf{E}^t \in \mathbb{R}^{n_1 n_2 \times d}$ denotes the embedding in t-th iteration, with d as the hidden size. At every iteration, the embedding is calculated by four hidden parts $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4 \in \mathbb{R}^{n_1 n_2 \times d}$. $\theta_1 \in \mathbb{R}^d$, $\theta_2 \in \mathbb{R}^{d \times d}$, $\theta_3 \in \mathbb{R}^d$, $\theta_4 \in \mathbb{R}^{d \times d}$ and $\theta_5 \in \mathbb{R}^d$ are the weight matrices in the neural networks. t is the index of the iteration and the total number of the iterations is T. We set the initial embedding $\mathbf{E}^0 = \mathbf{0}$ and use ReLU as the activation function.

As for the hidden parts, each hidden part represents a kind of feature: $\mathbf{h}_1$ is to calculate the impact of current permutation matrix $\mathbf{X}'$ which is transformed from the current partial solution $\mathbb{U}'$. $\mathbf{h}_2$ is to take neighbor's embedding into consideration, where $\mathbf{A}$ is the adjacency matrix of the association graph and divide $(n_1 - 1)(n_2 - 1)$ is for average since every vertex has $(n_1 - 1)(n_2 - 1)$ neighbors. $\mathbf{h}_3$ calculates the average of neighbor's vertex weights, where $\mathbf{F}$ is the vertex weight matrix. $\mathbf{h}_4$ is designed to extract the features of adjacent edges, where $\mathbf{W}$ is the edge weight matrix.

4.3.2 Q-Value Estimation Networks

Q-learning based algorithms use $\mathbf{Q}(s, a)$ to represent the value of taking action a in state s, as an expected value of the acquired reward after choosing this action. The agent picks the next action given the estimation of $\mathbf{Q}(s, a)$. The Q-value estimation network f_θ takes the embedding of the current state as input and predicts the Q-value for each possible action. We adopt Dueling DQN [46] as our function approximator to estimate the Q-value function. The architecture of our Q-value estimation network f_θ is:

$$
\begin{aligned}
\mathbf{h}_5 &= \text{ReLU}(\mathbf{E}^T \cdot \theta_6 + b_1) \\
\mathbf{h}_v &= \frac{\sum \mathbf{h}_5 \cdot \theta_7}{n_1 n_2} + b_2 \\
\mathbf{h}_a &= \mathbf{h}_5 \cdot \theta_8 + b_3 \\
\mathbf{Q} &= \mathbf{h}_v + \left(\mathbf{h}_a - \frac{\sum \mathbf{h}_a}{n_1 n_2}\right)
\end{aligned}
\qquad (8)
$$

where $\mathbf{E}^T$ is final output of the embedding network defined in Equation 7. $\mathbf{h}_5 \in \mathbb{R}^{n_1 n_2 \times d}$ is the hidden layer for embedding. $\mathbf{h}_v \in \mathbb{R}^1$ is the hidden layer for the state function. $\mathbf{h}_a \in \mathbb{R}^{n_1 n_2}$ is the hidden layer for the advantage function. $\theta_6 \in \mathbb{R}^{d \times d}, \theta_7 \in \mathbb{R}^d, \theta_8 \in \mathbb{R}^d$ are the weights of the neural networks. b_1, b_2, and b_3 are the bias vectors. $\mathbf{Q} \in \mathbb{R}^{n_1 n_2}$ is the final output of our Q-value estimate network, which predicts the value of every action given the current state.

The design of the state function and advantage function is to separate the value of state and action. Specifically, the state function predicts the value of different states and the advantage function predicts the value of each action given the particular state. We use $\mathbf{Q}(s, a; f_\theta)$ to denote the estimated Q-value by f_θ when the agent takes action a on state s. Note that the agent can only choose the vertex from the available vertices set $\mathbb{V}$. We add a mask on the output $\mathbf{Q}$ such as the values of the vertices not in $\mathbb{V}$ become minimal.

4.4. Experience Replay Memory

To improve the sample efficiency in reinforcement learning, we maintain a prioritized experience replay memory $\mathcal{M}$ [35] that stores the experience of the agent, defined as the transition (s_i, a_i, r_i, s_i') (denoting state, action, reward,

and state of next step respectively). As the training progresses, we add new transitions to $\mathcal{M}$ and remove old transitions. The agents will take samples from the experience replay memory to update their neural networks. We follow the idea of prioritized experience replay memory, which adds a priority for each transition and higher priority denotes higher probabilities to be sampled.

4.5. Model Updating

We follow the idea of Double DQN [15] to calculate the loss function and update the parameters. We pick the next action a' by the current Q-value estimate network f_θ, but use the target Q-value estimate network $f_{\theta-}$ to predict its value as Equation 9 shows. The motivation of designing this loss function is: the Q-value that is overestimated in one network will be mitigated to an extent in another network.

$$a' = \arg\max_{a'} \mathbf{Q}(s', a'; f_\theta) \tag{9}$$

$$\mathcal{L}(f_\theta; \mathcal{M}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{M}}\left[\left(r + \gamma \mathbf{Q}(s', a'; f_{\theta-}) - \mathbf{Q}(s, a; f_\theta)\right)^2\right]$$

where $\mathcal{L}(\cdot)$ is the loss function to be optimized. f_θ is the current Q-value estimation network and $f_{\theta-}$ is the target Q-value estimation network. The target network $f_{\theta-}$ shares the same architecture with f_θ and the parameters of $f_{\theta-}$ will be replaced by the parameters of f_θ every period of time. The design of such a target network is for keeping the target Q value remains unchanged for a period of time, which reduces the correlation between the current Q value and the target Q value and improves the training stability.

5. Experiments

5.1. Protocols

In the testing phase, given the affinity matrix $\mathbf{K}$, *RGM* predicts a permutation matrix $\mathbf{X}^{pred}$ transformed from its solution set $\mathbb{U}$. Based on $\mathbf{X}^{pred}$ and ground truth $\mathbf{X}^{gt}$, two evaluation metrics are used: objective score (also known as affinity score) and matching accuracy as defined by:

$$Score = \frac{\text{vec}(\mathbf{X}^{pred})^\top \mathbf{K} \text{ vec}(\mathbf{X}^{pred})}{\text{vec}(\mathbf{X}^{gt})^\top \mathbf{K} \text{ vec}(\mathbf{X}^{gt})}$$
$$Acc = \frac{\sum\left(\mathbf{X}^{pred} * \mathbf{X}^{gt}\right)}{n_1} \tag{10}$$

where $*$ denotes element-wise matrix multiplication.

Comparing *RGM* with existing supervised learning graph matching methods [51, 41, 13, 42] is unfair, because they require ground truth permutations as labels while *RGM* does not. *RGM* falls in line with learning-free graph matching solvers that use the same input. The following methods are compared in our experiments: **GAGM** [14] utilizes the graduated assignment technique, which can iteratively approximate the cost function by Taylor expansion. **RRWM** [8] proposes a random walk view of the

graph matching, with a re-weighted jump on graph matching. Given a continuous or discrete solution, **IFPF** [25] iteratively improves the solution via integer projection. **PSM** [12] improves the spectral algorithm through a probabilistic view. It presents a probabilistic interpretation of the spectral relaxation scheme. **GNCCP** [26] follows the convex-concave path-following algorithm. It provides a much simpler form of the partial permutation matrix. To deal with the singular point issue in the previous path following strategy, **BPF** [43] designs a branch switching technique to seek better paths at the singular points.

5.2. Experiments on Synthetic Dataset

We first evaluate *RGM* on the synthetic graphs generated following the protocol of [42]. We first generate sets of random points in the 2D plane. The coordinates of these points are sample from uniform distribution $U(0, 1) \times U(0, 1)$. First, we select 10 sets of points as the set of ground truth points. Then, we randomly scale their coordinates from $U(1 - \delta_s, 1 + \delta_s)$. The set of scaled points and the set of ground truth points are regarded as the pairwise graphs to be matched. We set there are 10 inliners without outlier, and δ_s varies from 0 to 0.5. For the calculation of affinity matrix $\mathbf{K}$, the node affinity is set by 0 and the edge affinity is set by the difference of edge length: $\mathbf{K}_{ia,jb} = exp\left(-\frac{(f_{ij}-f_{ab})^2}{\sigma_1}\right)$, where the f_{ij} is the edge length of E_{ij}. We generate 300 sets of scaled points for each ground truth sets and get 3,000 pairwise graphs in total. In all experiments, we split the data into the training set and testing set with the ratio of (2 : 1).

The results of the synthetic datasets are shown in Figure 2. Evaluations are performed in terms of the scale ratio δ_s. We can see that *RGM* performs the best in terms of matching accuracy and objective score in all experiments.

5.3. Experiments on Willow Object Dataset

Willow Object Dataset is collected from real images by [7]. It contains 256 images from 5 categories. For Each image, we use 10 distinctive key points with 2 randomly generated outliers. The experiments setup follows previous work [47, 18]. The affinity matrix $\mathbf{K}$ is calculated by $\mathbf{K}_{ia,jb} = \beta \cdot exp\left(-\frac{(f_{ij}-f_{ab})^2}{\sigma_2}\right) + (1-\beta) \cdot exp\left(-\frac{(\alpha_{ij}-\alpha_{ab})^2}{\sigma_3}\right)$, where f_{ij} denotes the edge length and α_{ij} denotes the edge angle. Evaluation is performed on all five categories. We also split the data into the training and testing by (2 : 1).

The results are shown in Table 1. We can see that *RGM* performs the best in four categories except for Winebottle. Since the difficulty of each category is not the same, the improvement of *RGM* varies in different categories. *RGM* can improve the matching accuracy in Car by more than 10%. Though *RGM* does not perform the best in Winebottle, the result of *RGM* is similar to the best baseline. Therefore, the average performance of *RGM* is still the best.

	Car		Duck		Face		Motorbike		Winebottle		Average		Inference Time
	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	
RRWM [8]	61.0%	0.936	69.3%	1.008	95.6%	0.968	74.6%	0.943	63.0%	0.956	72.7%	0.962	1.40%
GAGM [14]	59.7%	0.947	69.2%	1.006	95.5%	0.968	77.2%	0.965	**68.6%**	0.986	74.0%	0.974	6.00%
IPFP [25]	43.3%	0.832	43.0%	0.887	84.8%	0.945	53.5%	0.867	50.6%	0.915	55.0%	0.889	0.40%
PSM [12]	42.8%	0.757	44.1%	0.775	86.1%	0.890	69.0%	0.819	63.7%	0.828	61.1%	0.814	68.60%
GNCCP [26]	66.7%	0.958	65.9%	0.993	95.6%	0.968	76.5%	0.960	67.7%	0.980	74.5%	0.972	93.40%
BPF [43]	68.2%	0.980	72.1%	1.005	96.9%	0.969	80.3%	0.972	63.5%	**0.994**	76.2%	0.984	119.00%
RGM	**78.6%**	**1.013**	**76.4%**	**1.115**	**97.6%**	**0.984**	**85.3%**	**0.975**	66.1%	0.948	**80.8%**	**1.007**	100.00%

Table 1. Performance comparison w.r.t accuracy and objective score (the higher the better) in the Willow dataset. The last column shows the percentage of average inference time of all methods, compared to *RGM*. Following [18], all matching problems contain 10 inliers and 2 randomly generated outliers, and we construct the same affinity matrix for all methods by utilizing only the geometric information.
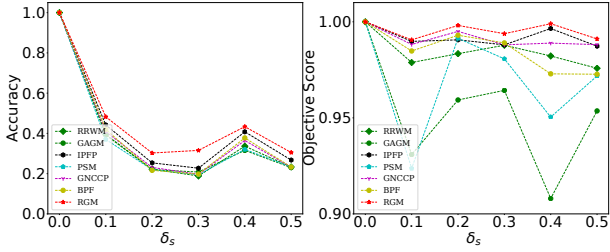

Figure 2. Performance comparison w.r.t objective score and accuracy (the higher the better) in the synthetic dataset.

	a	b	c	d	e	f	g	h	i	j
SM [24]	98	318	276	48	52	0	44	1292	54	22
RRWM [8]	80	294	204	44	50	0	52	1002	28	18
SK-JA [22]	100	304	266	58	44	0	52	1282	36	18
RGM	**72**	**292**	**178**	**24**	**34**	**0**	**36**	**996**	**18**	**14**
Upper Bound	68	292	160	16	28	0	26	996	14	8

Table 2. Performance comparison w.r.t objective score (the lower the better) with Esc16 instances of QAPLIB dataset.

We notice that the objective score of some methods is greater than 1, which means the objective score of the ground truth solution may not always be optimal. In the graph matching problem, we find that the mismatch between accuracy and objective score is quite common. We will discuss more about this in Section 5.7.

We admit that RL can be time-consuming. Since we split the training set and the testing set, we can record the testing time of *RGM*, which is comparable to the inference time of traditional solvers. Table 1 shows the average inference time of all methods, where the time consumption of *RGM* is similar to existing solvers GNCCP and BPF.

5.4. Experiments on Pascal VOC Object Dataset

Table 3 and Table 4 report results of our experiments on the Pascal VOC dataset. We also apply the train/test split rate mentioned before. We show the performance of every 20 class in this dataset, and we can see that *RGM* reaches the highest accuracy and objective score in most of the 20 classes, and outperforms all baselines on average.

Please note that the baselines in the experiments are learning-free methods, which use the same input information as *RGM* thus leads to a fair comparison. We admit that some of the existing deep learning based graph matching methods (GMN≈55% [51], PCA≈64% [41], NGM≈66% [42], LCS≈69% [44], BBGM>70% [32]) can reach higher average accuracy than *RGM* ≈59%. However, these learning-based methods must need labels (ground truth permutations) for supervised training, in other words, they require extra information compared to *RGM* and all baselines. **Therefore, it is unfair to compare *RGM* with existing deep graph matching methods.** Though unfair, we can see that *RGM*, the first reinforcement learning based GM method can perform close to the first deep learning based GM method GMN [51]. This performance is passable since our work is in a very early stage. As for the comparison with learning-free baselines, *RGM* does not use additional information, nor require ground-truth, and split the train/test set to prevent over-fitting. So, this comparison is relatively fair. Such a comparison setting is very common in the literature that utilizes reinforcement learning to solve the combinatorial optimization problems [6, 5, 19, 31, 20, 11, 2, 3, 29].

Besides, we show the generalization performance of RGM among similar categories in Figure 4. We use one class for training and another class for testing to evaluate the generalization ability of *RGM*. It turns out *RGM* does have generalization ability since it reaches good performance on similar classes. In this dataset, the objective score of some methods is also greater than 1, which means the mismatch between accuracy and objective score still happens.

5.5. Experiments on QAPLIB Dataset

We try to adapt our proposed solver *RGM* directly to real QAP instances. We conduct experiments on QAPLIB [4]. Due to its difficulty and complexity, we only choose one subset named "Esc16". "Esc16" stems from an application in the testing of self-testable sequential circuits, where the amount of additional hardware for the testing should be minimized. There are totally 10 available instances in

	Aero		Bike		Bird		Boat		Bottle		Bus		Car		Cat		Chair		Cow	
	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score
RRWM [8]	31.0%	1.021	24.7%	1.019	34.3%	1.020	37.3%	1.021	44.9%	1.016	68.7%	1.020	64.7%	1.020	46.5%	1.019	31.9%	1.020	36.6%	1.018
GAGM [14]	37.6%	1.021	34.4%	1.021	49.3%	1.021	42.8%	1.017	60.4%	1.020	75.0%	1.020	62.3%	1.020	37.5%	1.024	37.5%	1.021	47.8%	1.023
IPFP [25]	25.8%	1.021	23.4%	1.020	31.4%	1.020	36.2%	1.021	38.4%	1.017	55.6%	1.020	52.5%	1.020	38.3%	1.017	28.9%	1.021	31.2%	1.018
PSM [12]	40.2%	1.025	44.1%	1.026	**58.7%**	1.021	62.6%	1.024	56.2%	1.018	40.7%	1.021	58.9%	1.021	41.8%	1.020	43.8%	1.023	42.7%	1.021
GNCCP [26]	32.3%	1.021	37.0%	1.021	36.8%	1.020	45.6%	1.021	41.1%	1.016	52.5%	1.017	58.2%	1.019	51.0%	1.022	36.0%	1.023	45.3%	1.022
BPF [43]	38.1%	1.022	**47.5%**	1.025	55.6%	1.022	**64.1%**	1.024	59.8%	1.012	41.3%	1.021	54.9%	1.021	42.7%	1.022	**48.1%**	**1.024**	45.8%	1.022
GMN [51]	31.9%	-	47.2%	-	51.9%	-	40.8%	-	68.7%	-	72.2%	-	53.6%	-	52.8%	-	34.6%	-	48.6%	-
RGM	**39.3%**	**1.055**	43.5%	**1.032**	55.0%	**1.029**	58.0%	**1.027**	**71.4%**	**1.022**	**76.0%**	**1.024**	**68.3%**	1.021	**61.0%**	**1.026**	35.2%	1.009	**52.1%**	**1.023**

Table 3. Performance comparison w.r.t accuracy and objective score (the higher the better) in the Pascal VOC dataset.

	Table		Dog		Horse		Mbike		Person		Plant		Sheep		Sofa		Train		Tv		**Average**	
	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score	Acc	Score
RRWM [8]	32.9%	0.999	36.3%	1.018	33.9%	0.995	41.1%	1.019	23.0%	1.019	45.4%	1.019	50.2%	1.019	46.3%	1.021	73.9%	1.022	84.5%	0.999	48.6%	1.017
GAGM [14]	40.5%	1.000	49.8%	1.021	43.3%	1.000	49.3%	1.014	26.2%	1.019	67.0%	1.022	52.0%	1.021	46.8%	1.016	86.2%	0.999			53.5%	1.019
IPFP [25]	36.8%	1.000	32.5%	1.018	28.7%	0.996	31.5%	1.016	23.3%	1.021	33.9%	1.019	38.4%	1.018	32.5%	1.021	73.1%	1.021	70.7%	0.999	41.9%	1.017
PSM [12]	23.1%	0.994	38.5%	1.020	42.4%	1.000	43.6%	1.001	45.3%	1.022	73.3%	1.023	49.1%	1.020	**63.9%**	**1.023**	85.4%	1.021	40.8%	0.997	53.7%	1.019
GNCCP [26]	33.2%	0.999	44.7%	1.019	36.4%	0.998	41.4%	1.017	31.7%	1.021	54.1%	1.019	50.6%	1.019	47.5%	1.021	80.6%	1.021	78.6%	0.998	54.7%	1.018
BPF [43]	16.3%	0.991	40.9%	1.018	39.8%	0.999	46.6%	1.018	**46.2%**	1.022	74.3%	1.024	44.6%	1.017	59.1%	1.022	82.9%	1.022	41.3%	0.997	54.7%	1.020
GMN [51]	72.3%	-	47.7%	-	54.8%	-	51.0%	-	38.6%	-	75.1%	-	49.5%	-	45.0%	-	83.0%	-	86.3%	-	55.3%	-
RGM	**44.0%**	**1.002**	**55.2%**	**1.021**	**53.3%**	**1.014**	**55.7%**	**1.018**	36.6%	**1.032**	**81.6%**	**1.024**	**52.4%**	**1.022**	57.0%	**1.023**	**88.7%**	**1.022**	**87.6%**	**1.000**	**58.6%**	**1.024**

Table 4. Continued performance comparison w.r.t accuracy and objective score (the higher the better) in the Pascal VOC dataset. Please note that it is unfair to compare *RGM* with existing deep graph matching methods, since they require ground truth permutations as labels while *RGM* does not. The baselines listed in the table use the same input as *RGM* (except GMN [51] that learns the affinity), which leads to a direct comparison under exactly the same setting. Interestingly, *RGM*, the first RL based GM method performs close to the first deep learning based GM method GMN. The accuracy of GMN is quoted from [51, 44] where the affinity score is unavailable.
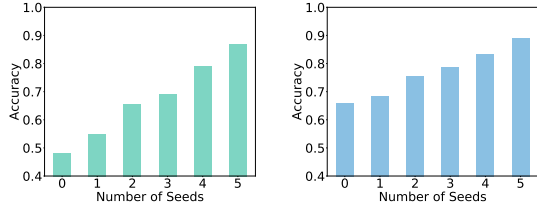
(a) Synthetic Dataset   (b) Willow Object Dataset

Figure 3. Case Study: performance comparison w.r.t accuracy (the higher the better) in terms of the different number of initial seeds.

Testing Instance

	Car	Bus	Chair	Sofa	Dog	Cat	Train	Tv
Car	68.3%	63.2%	25.2%	37.8%	38.5%	46.3%	60.9%	47.6%
Chair	33.6%	49.0%	35.2%	49.1%	37.2%	44.9%	57.4%	36.9%
Dog	47.8%	56.1%	28.3%	45.3%	55.2%	54.9%	61.0%	71.4%
Train	51.0%	68.0%	30.3%	44.1%	41.3%	52.9%	88.7%	83.0%

(Training Instance — row indices)

Figure 4. Generalization analysis of RGM w.r.t accuracy. Row indices are training classes and column indices are testing classes.

"Esc16". We simply use the first 5 instances for training and the last 5 instances for testing. The results are shown in Table 2. We compare with two existing solvers: SM [24] solves the QAP by the spectral numerical technique; Sinkhorn-JA [22] solves the lifted linear program relaxations of QAP. "Upper Bound" denotes the objective score of the proved optimal solution. We can see that *RGM* outperforms both training and testing in this category of QAPLIB, and reaches the upper bound in three instances.

5.6. Case Study: Seeded Graph Matching

As we mentioned before, our proposed solver *RGM* is to learn the back-end decision making process of the graph matching problem. More importantly, *RGM* can take supplementary information for graph matching, such as initial seeds. Initial seed means that one node in each of the original pairwise graphs is already matched by human or other information sources. However, existing learning based graph matching methods can not utilize these valuable initial seeds because they provide the permutation matrix solution in one shot, not like *RGM* solves the problem progressively. Figure 3 shows the results of this case study.

We can see that adding suitable initial seeds does improve the performance. It shows the importance of extra initial seeds ignored by previous deep graph matching methods.

5.7. Discussions and Limitations of back-end Solver

Let us re-emphasize an important statement that graph matching problem consists of two parts: extracting features from images to formulate the QAP instance (front-end) and solving the underlying QAP (back-end). Existing graph matching learning models [41, 51] mostly focus on learning the front-end feature extractor by CNN and GNN, and fixed learning-free solvers [24, 8, 9] are adopted as the back-end. In contrast, *RGM* is focused on learning the back-end solver without touching the front-end feature learner. It gives some interpretations in several folds worth noting:

First, *RGM* cannot be directly compared with existing learning based methods as they focus on different parts of the learning pipeline. And how to integrate these front-end and back-end components in an end-to-end learning fashion is left for future work as an open question. Besides, the training process of *RGM* is label-free. Second, our method can take either the non-learnable features or learned features in our affinity score maximization pipeline. **Such a**

Affinity score	[0.00, 0.99)	[0.99, 1.00)	= 1.00	(1.00, +∞)
Proportion	4.0%	11.6%	45.0%	39.4%
Accuracy	40.6%	61.2%	100.0%	56.5%

Table 5. Distribution of the affinity score and accuracy of the matching solutions found by *RGM* on bus category in Pascal VOC dataset, where the affinity matrix is built from a pretrained feature extractor [42]. The mismatch of accuracy and score is clear.

pipeline implies a common assumption that the highest score corresponds to the ground truth matching. However, whatever the feature is extracted by certain rules like Gaussian kernel [18] or by CNN/GNN [41, 42], such an assumption does not necessarily hold, as shown in Table 5. It turns out the learned affinity function is biased from the true accuracy, making the back-end solver *RGM* struggle, as *RGM* is purely based on reaching a higher score. Results from Table 1&3&4 also show such inconsistency between the matching accuracy and the learned affinity function by front-end, suggesting the limitation of existing deep graph matching models. It indicates the performance of existing deep graph matching methods highly relies on the supervision of ground truth labels rather than actually solving the QAP, which is the core of the graph matching problem.

6. Conclusion and Outlook

In this paper, for the first time to our knowledge, the classic graph matching problem is formulated and successively solved as a reinforcement learning task, by proposing a learning solver *RGM*. It uses GNN based embedding networks to calculate the representation before conducting the Q-learning. Extensive experiments on both synthetic and real-world datasets show the cost-effectiveness of *RGM*, of which the inference time is close to existing solvers.

References

[1] Yunsheng Bai, Derek Xu, Alex Wang, Ken Gu, Xueqing Wu, Agustin Marinovic, Christopher Ro, Yizhou Sun, and Wei Wang. Fast detection of maximum common subgraph via deep q-learning. *arXiv preprint arXiv:2002.03129*, 2020. 2

[2] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, abs/1611.09940, 2017. 2, 7

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020. 2, 7

[4] R. Burkard, S. Karisch, and F. Rendl. Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997. 7

[5] Quentin Cappart, Thierry Moisan, L. Rousseau, Isabeau Pr'emont-Schwarz, and André Ciré. Combining reinforcement learning and constraint programming for combinatorial optimization. *ArXiv*, abs/2006.01610, 2020. 2, 7

[6] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, 2019. 2, 7

[7] Minsu Cho, Alahari Karteek, and J. Ponce. Learning graphs to match. *2013 IEEE International Conference on Computer Vision*, pages 25–32, 2013. 1, 6

[8] Minsu Cho, J. Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *ECCV*, 2010. 2, 6, 7, 8

[9] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *NIPS*, pages 2292–2300, 2013. 2, 8

[10] Hanjun Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016. 5

[11] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda L Dietrich, David P. Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. *ArXiv*, abs/2006.03750, 2020. 2, 7

[12] A. Egozi, Y. Keller, and H. Guterman. A probabilistic approach to spectral graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:18–27, 2013. 6, 7, 8

[13] Matthias Fey, Jan E Lenssen, Christopher Morris, Jonathan Masci, and Nils M Kriege. Deep graph matching consensus. In *Int. Conf. Learn. Represent.*, 2020. 1, 6

[14] S. Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18:377–388, 1996. 2, 6, 7, 8

[15] H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016. 2, 4, 6

[16] Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, W. Dabney, Dan Horgan, B. Piot, Mohammad Gheshlaghi Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018. 2, 4

[17] Adel Hlaoui and Shengrui Wang. A new algorithm for graph matching with application to content-based image retrieval. In *SSPR/SPR*, 2002. 1

[18] Zetian Jiang, Tianzhe Wang, and Junchi Yan. Unifying offline and online multi-graph matching via finding shortest paths on supergraph. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 6, 7, 9

[19] Chaitanya K. Joshi, Quentin Cappart, L. Rousseau, Thomas Laurent, and Xavier Bresson. Learning tsp requires rethinking generalization. *ArXiv*, abs/2006.07054, 2020. 2, 7

[20] Elias Boutros Khalil, Hanjun Dai, Yuyu Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *ArXiv*, abs/1704.01665, 2017. 2, 7

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Int. Conf. Learn. Represent.*, 2017. 5

[22] Y. Kushinsky, Haggai Maron, Nadav Dym, and Y. Lipman. Sinkhorn algorithm for lifted assignment problems. *SIAM J. Imaging Sci.*, 12:716–735, 2019. 7, 8

[23] Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963. 1, 3

[24] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, 2:1482–1489 Vol. 2, 2005. 2, 3, 7, 8

[25] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In *NIPS*, 2009. 6, 7, 8

[26] Zhi-Yong Liu, Hong Qiao, and Lei Xu. An extended path following algorithm for graph-matching problem. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1451–1456, 2012. 6, 7, 8

[27] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *EJOR*, pages 657–90, 2007. 2

[28] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016. 2

[29] Nina Mazyavkina, S. Sviridov, S. Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *ArXiv*, abs/2003.03600, 2020. 2, 7

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2, 4

[31] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018. 2, 7

[32] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pages 407–424. Springer, 2020. 2, 7

[33] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. *arXiv preprint arXiv:2003.11657*, 2020. 2

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Trans. on Neural Networks*, 2009. 5

[35] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 2, 5

[36] Christian Schellewald and Christoph Schnörr. Probabilistic subgraph matching based on convex relaxation. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 171–186. Springer, 2005. 2

[37] Yang Shen, Weiyao Lin, Junchi Yan, Mingliang Xu, Jianxin Wu, and Jingdong Wu. Person re-identification with correspondence structure learning. In *ICCV*, 2015. 1

[38] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 2

[39] R. Sutton and A. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005. 4

[40] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *ArXiv*, abs/1506.03134, 2015. 3

[41] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Int. Conf. Comput. Vis.*, 2019. 1, 2, 6, 7, 8, 9

[42] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *ArXiv*, abs/1911.11308, 2019. 2, 3, 6, 7, 9

[43] Tao Wang, Haibin Ling, Congyan Lang, and Songhe Feng. Graph matching with adaptive and branching path following. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017. 6, 7, 8

[44] Tao Wang, He Liu, Yidong Li, Yi Jin, Xiaohui Hou, and Haibin Ling. Learning combinatorial solver for graph matching. In *CVPR*, pages 7568–7577, 2020. 2, 7, 8

[45] Yansheng Wang, Yongxin Tong, Cheng Long, Pan Xu, Ke Xu, and Weifeng Lv. Adaptive dynamic bipartite graph matching: A reinforcement learning approach. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1478–1489. IEEE, 2019. 2

[46] Ziyu Wang, T. Schaul, Matteo Hessel, H. V. Hasselt, Marc Lanctot, and N. D. Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016. 2, 4, 5

[47] J. Yan, M. Cho, H. Zha, X. Yang, and S. Chu. Multi-graph matching via affinity optimization with graduated consistency regularization. *TPAMI*, 2016. 6

[48] Junchi Yan, Shuang Yang, and Edwin Hancock. Learning graph matching and related combinatorial optimization problems. In *IJCAI*, 2020. 2

[49] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *ICMR*, 2016. 2

[50] Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In *Int. Conf. Learn. Represent.*, 2020. 1, 2

[51] Andrei Zanfir and Christian Sminchisescu. Deep learning of graph matching. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 1, 2, 6, 7, 8

[52] Zhen Zhang and Wee Sun Lee. Deep graphical feature learning for the feature matching problem. In *ICCV*, pages 5087–5096, 2019. 1

[53] Feng Zhou and Fernando De la Torre. Factorized graph matching. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2012. 2

[54] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and M. Sun. Graph neural networks: A review of methods and applications. *ArXiv*, abs/1812.08434, 2018. 3

[55] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. Causal discovery with reinforcement learning. *arXiv preprint arXiv:1906.04477*, 2019. 2