Learning Combinatorial Solver for Graph Matching

Tao Wang[1,2] He Liu[1] Yidong Li[1] Yi Jin[1] Xiaohui Hou[2] Haibin Ling[3]
[1]The Beijing Key Laboratory of Traffic Data Analysis and Mining, Beijing Jiaotong University, Beijing 100044, China
[2]HiScene Information Technologies, Shanghai 201210, China
[3]Stony Brook University, Stony Brook, NY 11794, USA.
{twang,liuhe1996,ydli,yjin}@bjtu.edu.cn, houxh@hiscene.com, hling@cs.stonybrook.edu

Abstract

Learning-based approaches to graph matching have been developed and explored for more than a decade, and have grown rapidly in scope and popularity recently. However, previous learning-based algorithms, with or without deep learning strategy, mainly focus on the learning of node and/or edge affinities generation, and pay less attention to the learning of the combinatorial solver. In this paper we propose a fully trainable framework for graph matching, in which learning of affinities and solving for combinatorial optimization are not explicitly separated as in many previous arts. We firstly convert the problem of building node correspondences between two input graphs to the problem of selecting reliable nodes from a constructed assignment graph. Subsequently, the graph network block module is adopted to perform computation on the graph to form structured representations for each node. It finally predicts a label for each node that is used for node classification, and the training is performed under the regularization of both permutation differences and the one-to-one matching constraints. The proposed method is evaluated on four public benchmarks in comparison with state-of-the-art algorithms, and the experimental results illustrate its excellent performance.

1. Introduction

As a fundamental problem in computer science, graph matching involves establishing correspondences between vertex sets of given graphs while keeping the consistency between their edge sets as well. It closely relates to many computer vision problems including feature registration [23], shape reconstruction [30], object recognition [31], visual tracking [28], object labeling [22], etc. Graph matching is a well-known general NP-hard problem, and it is hard to acquire a global optimum for graphs of reasonable sizes. Consequently, approximate algorithms that seek acceptable suboptimal solutions by utilizing relaxation to harness the solution searching are popular in this topic. Despite decades of research effort devoted to graph matching [6,8,10,11,13,20,29,34,35,38], it remains a challenging problem due to not only the non-convex objective but also the combinatorial constraints.

Aiming to improve the matching accuracy in the assistance of data analysis and machine learning technology, learning-based algorithms have been investigated in the past decade. Early methods [4, 16, 17] employ simple and shallow parametric models that control geometric affinities between pairs of vertices, and the promotion to the matching performance derived from these models are usually limited. With the growing interest in using deep features for both geometric and semantic visual matching tasks, learning graph matching using deep network has attracted much research attention. Researchers [27, 33] presented deep learning frameworks for graph matching with general applicability to model deep feature extraction, unary and pairwise affinity generation and combinatorial optimization. However, these studies mainly focus on the architecture for learning the affinity functions while less investigation has been devoted to the design of the combinatorial solver. In particular, in these studies the quadratic assignment problem is usually relaxed to easier ones and is solved by some non-parametric and relatively simple combinatorial solvers that may hurt the matching accuracy. It therefore demands research attention on how to design fully trainable network to boost graph matching.

Addressing the above mentioned issues, we propose a novel deep learning framework for graph matching aiming to improve the matching accuracy. Roughly speaking, our framework is a fully trainable network designed on top of *graph neural network*, in which learning of affinities and solving for combinatorial optimization are not explicitly separated. We first construct an assignment graph for two input graphs to be matched considering each candidate match a node, and convert the problem of building node correspondences between input graphs to the problem of selecting reliable nodes from the constructed assignment graph. Subsequently, the graph network block module is extended in our framework to perform computation on the constructed assignment graph, which forms structured representations for each node by several convolutional filters. Our network finally reads out node labels from the updated graph state, which are used to classify the nodes into positive or negative ones. Moreover, the training of our network is guided by taking both the permutation differences and the one-to-one matching constraints as supervision.

For evaluation we test our algorithm on four public benchmarks, in comparison with eight state-of-the-art baselines including both non-learning and learning-based algorithms. Our algorithm exhibits robustness against noises and outliers, and outperforms all baseline algorithms in general. The source code along with the experiments is made available at `https://www.cs.stonybrook.edu/~hling/code/LearningGraphMatching.zip`.

In summary, with the proposed learning framework for graph matching, this paper makes contribution in three-fold:

- we transform learning of graph matching to learning of node selection by constructing an assignment graph given two input graphs to be matched;
- we combine learning of affinities and solving for combinatorial optimization into a unified learning framework, and extend the graph network block module for structural representation and relational reasoning; and
- we design a novel loss function in which the one-to-one matching constraints are imposed to supervise the training of the network.

2. Related Work

2.1. Traditional Graph Matching

Graph matching has been investigated for decades and many algorithms have been proposed. In general, graph matching has combinatorial nature that makes the global optimum solution hardly available, and approximate solutions that seek acceptable suboptimal solutions are thus commonly applied to graph matching.

A notable work by Leordeanu and Hebert [13] approximates graph matching based on spectral relaxation, which constructs an assignment graph by representing the potential correspondences as nodes and between-correspondence pairwise agreements as edges. The discovery of correct correspondences is formulated as detecting the main strongly connected cluster in the assignment graph, and is then solved using an eigen-analysis approach. This work is later extended by Cour et al. [8], who encode the mapping constraints into the spectral decomposition and apply a bistochastic normalization on the compatibility matrix to considerably reduce matching errors; and by Cho et al. [6], who cast graph correspondence searching as a node ranking and selection problem, and introduce an affinity-preserving random walk algorithm to drive the node ranking based on its quasi-stationary distribution. From a different and probabilistic perspective, Zass and Shashua [35] propose a soft matching criterion for the problem input and output, and derive an algebraic relation between the hyper-edge weight matrix and the desired node-to-node probabilistic matching. Later, Egozi et al. [10] combine the spectral relaxation solution and the probabilistic framework by interpreting the former as a maximum likelihood estimate of the assignment probabilities, and present a new probabilistic formulation of quadratic matching by relaxing some of the assumptions.

Since constraint relaxation and post discretization are commonly used in the approximate methods, many important algorithms have been designed on top of the relaxation and discretization strategy. For example, Gold and Rangarajan [11] employ the graduated assignment technique to iteratively solve a series of linear approximations of the cost function using Taylor expansion. Along the line there comes the group of so called *path following* algorithms. In particular, Zaslavskiy et al. [34] reformulate graph matching as a *convex-concave relaxation procedure* (CCRP) problem and then solve it by interpolating between two relatively simpler and relaxed problems, one with convex relaxation and the other with concave relaxation. Following the work, Zhou and De la Torre [38] factorize an affinity matrix into a Kronecker product of smaller matrices and provide specified convex relaxation and concave relaxation based on the factorized matrices; Liu and Qiao [20] propose the *graduated nonconvexity and concavity procedure* (GNCCP) and prove that it equivalently realizes CCRP on partial permutation matrix; Wang et al. [29] present the path following algorithm that uses *adaptive and branching path following* (ABPF) strategy to discover singular points on the original path and then branches new paths for potential better results.

Different from the above algorithms that drop the discrete constraints for optimization, some purely discrete methods [1, 32] are recently proposed to search the solution directly in the discrete space. These methods do not require any post-optimization step because the generated solutions always obey the discrete affine mapping constraints, but they usually suffer from limited optimality, which is dependent largely on the initialization [1].

2.2. Learning Graph Matching

The above studies mostly rely on handcrafted affinities, which are taken as input of the combinatorial solver. Such a predefined parametric affinity model will limit the flexibility to capture the structure of a real-world matching task, and inappropriate affinity model may make the matching solver deviating from the ground truth matching solution.

Addressing this issue, the learning for graph matching has demonstrated its superior performance in terms of improving matching accuracy [36, 37], which is mainly boosted by learning the parameters of the graph affinity metric to replace the handcrafted affinity metric. Most conventional algorithms for learning graph matching are supervised ones [4, 14, 24] that require detailed labeling of each node correspondence in each positive graph for training. These algorithms use large margin methods [25], nonlinear inverse optimization [19], and smoothing-based techniques to train matching parameters in a supervised fashion, respectively. Compared to supervised methods, the unsupervised method [16] does not require a large amount of node-level labeling. Later, Leordeanu et al. [17] offer a semi-supervised learning formulation for models with constraints beyond second order. Different from these methods, Cho [5] proposes to parameterize a graph model for all instances of a class and learn its structural attributes for visual object matching.

The literature on deep learning of graph matching remains limited, despite the demonstrated power of deep learning techniques in many areas. The few pioneering studies mainly aimed to encode parametric affinity functions in deep networks, so as to obtain correct matching assignments under computed node and edge affinities. Zanfir and Sminchisescu [33] formulates graph matching as a quadratic assignment problem under unary and pair-wise node affinities represented using deep parametric feature hierarchies. It adopts spectral matching [13] as the combinatorial solver which is differentiable for back propagation. Wang et al. [27] employ the *graph convolutional network* (GCN) framework [12] as a node embedding module that aggregates graph structure information to generate node-to-node affinity. By this way, graph matching is relaxed to linear assignment and a Sinkhorn net is adopted as the combinatorial solver.

Our work falls into the group of deep learning algorithms. Compared with previous arts, our method focuses on learning of not only the affinity functions but also the combinatorial solver, which are effectively combined into a fully trainable graph network. Strong structured representations and their relational inductive biases are imposed into our learning framework to promote the matching accuracy, and the experiments illustrate its excellent performance.

3. Problem Formulation

3.1. The Graph Matching Problem

An attributed relational graph of n nodes can be represented by $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathcal{V}, \mathcal{E})$, where

- $\mathbb{V} = \{v_1, \dots, v_n\}$ denotes the node set,
- $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ denotes the edge set,
- $\mathcal{V} = \{\mathbf{v}_i | \mathbf{v}_i \in \mathbb{R}^{d_V}, i = 1, 2, ...n\}$ denotes the node attribute set, and
- $\mathcal{E} = \{\mathbf{e}_i | \mathbf{e}_i \in \mathbb{R}^{d_E}, i = 1, 2, ..., |\mathbb{E}|\}$ denotes the edge attribute set.

The node relations of an undirect graph are often conveniently represented by a symmetric adjacency matrix $A \in \mathbb{R}^{n\times n}$, such that $A_{ij} = 1$ if and only if there is an edge $(v_i, v_j) \in \mathbb{E}$.

For graph matching, given two graphs $\mathbb{G}^{(i)} = (\mathbb{V}^{(i)}, \mathbb{E}^{(i)}, \mathcal{V}^{(i)}, \mathcal{E}^{(i)})$ of n node[1], $i = 1, 2$, the problem is to find a node correspondence $X \in \{0,1\}^{n\times n}$ between $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$, in which each element $X_{ij} = 1$ if and only if $v_i^{(1)} \in \mathbb{V}^{(1)}$ corresponds to $v_j^{(2)} \in \mathbb{V}^{(2)}$. In practice, the matching is often restricted to be a one-to-one function that requires $X\mathbf{1}_n = \mathbf{1}_n$ and $X^\top \mathbf{1}_n = \mathbf{1}_n$, where $\mathbf{1}_n$ denotes a vector of n ones. The graph matching problem can be formulated as a *quadratic assignment problem* (QAP)

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \mathbf{x}^\top K \mathbf{x}, \tag{1}$$

where $\mathbf{x} \doteq \mathrm{vec}(X) \in \{0,1\}^{n^2}$ is the vectorized version of matrix X and $K \in \mathbb{R}^{n^2\times n^2}$ is the corresponding affinity matrix. Specifically, K is defined to measure pairwise affinities

$$K_{\mathrm{ind}(i_1,i_2)\mathrm{ind}(j_1,j_2)} = \begin{cases} c_{i_1 i_2} & \text{if } i_1 = j_1 \text{ and } i_2 = j_2, \\ d_{i_1 j_1 i_2 j_2} & \text{else if } A^{(1)}_{i_1 j_1} A^{(2)}_{i_2 j_2} > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathrm{ind}(\cdot,\cdot)$ is a bijection mapping a node correspondence to an integer index, $c_{i_1 i_2}$ measures the consistency between the node attributes $\mathbf{v}^{(1)}_{i_1}$ and $\mathbf{v}^{(2)}_{i_2}$, and $d_{i_1 j_1 i_2 j_2}$ the the consistency between edge attributes $\mathbf{e}^{(1)}_{(i_1,j_1)}$ and $\mathbf{e}^{(2)}_{(i_2,j_2)}$.

3.2. Matching as Node Labeling

Different from the traditional graph matching algorithms [6, 8, 10, 11, 13, 20, 29, 34, 35, 38] that usually adopt handcrafted affinity functions c and d, previous learning-based algorithms [4, 16, 27, 33] are devoted to learning appropriate affinity functions from data. However, the combinatorial solvers for the QAP (Eq. 1) employed in these learning algorithms are still un-trainable and relatively simple, which may hurt the final matching accuracy. In this paper we propose a novel learning framework that integrates the learning of both affinity functions and the combinatorial solver into a unified node labeling pipeline.

Basically, the problem of graph matching between the two graphs $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$ can be interpreted in a node labeling view by constructing an *assignment graph* $\mathbb{G}^A = (\mathbb{V}^A, \mathbb{E}^A, \mathcal{V}^A, \mathcal{E}^A)$ following [6, 13]. Given two graphs $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$, we consider each candidate correspondence

[1]For simplicity, we assume the two graphs to be matched have the same size n, and formulations can be easily extended to handle varied sizes, *e.g.*, by adding dummy nodes.

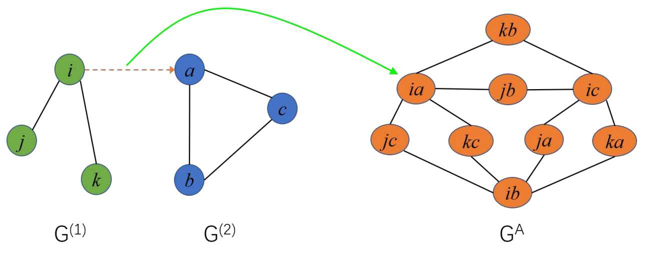


Figure 1. Illustration of assignment graph.

$(v_i^{(1)}, v_a^{(2)}) \in \mathbb{V}^{(1)} \times \mathbb{V}^{(2)}$ a node $v_{ia}^A \in \mathbb{V}^A$. For edge generation of $\mathbb{G}^A$, we build an edge between a pair of nodes $v_{ia}^A, v_{jb}^A \in \mathbb{V}^A$ if and only if there are two edges $(v_i^{(1)}, v_j^{(1)}) \in \mathbb{E}^{(1)}$ and $(v_a^{(2)}, v_b^{(2)}) \in \mathbb{E}^{(2)}$.

Fig. 1 illustrates a sampled assignment graph where each candidate match between $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$ corresponds to a node in the assignment graph $\mathbb{G}^A$. Thus, the original graph matching problem between $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$ is converted to classifying the nodes in $\mathbb{G}^A$ into positive nodes and negative ones. For example, building node correspondences $\{(v_i^{(1)}, v_a^{(2)}), (v_j^{(1)}, v_c^{(2)}), (v_k^{(1)}, v_b^{(2)})\}$ between $\mathbb{V}^{(1)}$ and $\mathbb{V}^{(2)}$ is equivalent to selecting nodes $\{v_{ia}^A, v_{jc}^A, v_{kb}^A\}$ out of $\mathbb{V}^A$.

In previous approaches to this node selection problem, node attributes $\mathcal{V}^A$ and edge attributes $\mathcal{E}^A$ are usually assigned according to the predefined affinity functions, and some optimization techniques, for instance eigenanalysis [13] and Markov random walks [6], are subsequently employed to acquire reliable nodes from the assignment graph $\mathbb{G}^A$.

Different from these hand-engineered algorithms, our algorithm does not require predefined affinity functions. It instead combines the raw attributes of $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$ to form the initial attributes of $\mathbb{G}^A$ as

$$\begin{aligned} \mathbf{v}_{ia}^A &= [\mathbf{v}_i^{(1)}; \mathbf{v}_a^{(2)}], \\ \mathbf{e}_{(ia,jb)}^A &= [\mathbf{e}_{(i,j)}^{(1)}; \mathbf{e}_{(a,b)}^{(2)}]. \end{aligned} \tag{3}$$

In order to learn how to select reliable nodes from the generated assignment graph $\mathbb{G}^A$, we propose a fully trainable graph network that takes $\mathbb{G}^A$ as input, performs computation over the structure, and predicts a label for each node as output. The predicted label of each node, for instance v_{ia}^A, is used to distinguish whether the candidate match $(v_i^{(1)}, v_a^{(2)})$ is a desired one for the original graph matching problem. The pipeline of the proposed learning framework is described in details in Sec. 4.

4. The Proposed Method

As described in Sec. 3.2, the graph matching problem is transformed to node labeling on the constructed assignment graph. Although the input graphs are usually extracted from images (for example in our experiments), the traditional *convolutional neural network* (CNN) is inadequate to perform computation and reasoning on graphs due to lacks of relational inductive biases.

Graph neural networks (GNNs), which operate on graphs and structure their computations accordingly, have grown rapidly in scope and popularity in recent years [2, 9, 12, 18, 26]. Inspired by these studies, we explore a flexible learning-based approach to graph matching based on GNN, which embeds strong relational inductive biases to capitalize on explicitly structured representations and computations.

4.1. Pipeline Overview

Our learning framework for graph matching is designed on the top of *graph network block* (GNB) presented in [2], which defines a class of functions for relational reasoning over graph-structured representations. It is extended in our framework to fit the graph matching problem by eliminating redundant components and redefining certain functions. We employ the extended module to perform computation over the constructed assignment graph and return the predicted node labels as output for node classification.

As illustrated in Fig. 2, the proposed learning pipeline for graph matching consists of four main modules:

- **Assignment Graph Generation:** This module takes the graphs to be matched as input, and constructs an assignment graph by considering each candidate match as a node in the graph, as that is described in Sec. 3.2.
- **Encoder/Decoder:** The encoder module takes the constructed assignment graph as input, and transforms its attributes into latent representations. Decoder is coupled to the encoder module, which reads out the final output from the updated graph state and reports the predicted node labels for loss computation.
- **Convolution Module:** This module is the core component of our learning framework, which convolves the information of neighborhoods of each node to form a structured representation through several convolutional operators. Each instance of the convolution module consists of an edge convolution layer and a node convolution layer, and k instances are stacked to aggregate the information of k^{th}-order neighborhoods, *i.e.*, the nodes that are at maximum k steps away from the central node.
- **Loss Computation:** In addition to the permutation loss that evaluates the difference between the predicted node labels and the ground-truth node labels, we impose the one-to-one matching constraints into the loss function to guide the training of the network.

The proposed framework takes raw attributes of the nodes and edges as input and predicts node labels based on

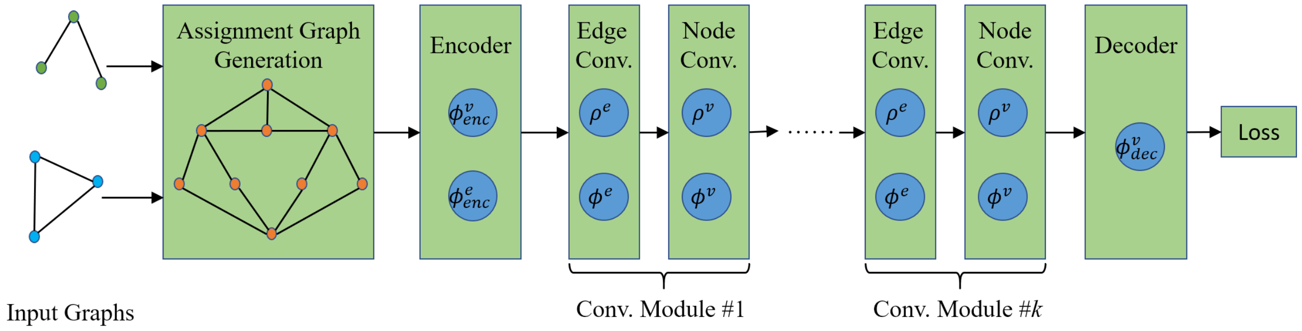


Figure 2. The pipeline overview of the proposed learning graph matching framework. We first construct an assignment graph given a pair of input graphs to be matched. The encoder module transforms the attributes of the constructed assignment graph into latent representations by two parametric update functions ϕ^v_{enc} and ϕ^e_{enc}. Subsequently, the stacked k instances of the convolution module convolves the k^{th}-order neighborhoods of each node to form its structured representations via four convolutional filters ρ^e, ϕ^e,ρ^v and ϕ^v . The convolved node attributes are finally decoded to node labels by an update function ϕ^v_{dec}.

convolved node features. In this pipeline the learning of affinities and solving for combinatorial optimization are intertwined and iterated through stacked multiple instances of the convolutional module. Our network finally produce a soft matching, and we adopt a simple greedy/max-activation approach as a post-processing for testing to guarantee a combinatorial one-to-one matching.

It is notable that the generation of input graphs, including node and edge feature extraction and edge generation, is dependent on the task and is not embedded as a component into our network. Different from [27, 33] that employ CNN to extract visual features from images, we ignore all appearance information and utilize only geometric cues. Specifically, we take the coordinates of each point as the node feature, and concatenate the features of two nodes associated with the same edge to form the edge feature. The reason of ignoring appearance cues lies that our framework is not limited in the scenario of matching images despite it is validated on two real image datasets in experiments. Furthermore, eliminating such a CNN module makes our framework more lightweight and easier to train. In experiments our algorithm exhibits excellent performance and outperforms state-of-the-art baselines despite of absence of appearance cues.

4.2. Encoder and Decoder

The encoder module transforms the attributes of the input graph into latent representations by two parametric update functions ϕ^v_{enc} and ϕ^e_{enc}. In this work ϕ^v_{enc} and ϕ^e_{enc} are designed as two *multi-layer perceptrons* (MLPs), each of which takes respectively a node attribute vector and an edge attribute vector as input and transforms them into latent spaces. Note that ϕ^v_{enc} is mapped across all nodes to compute per-node updates, and ϕ^e_{enc} is mapped across all edges to compute per-edge updates.

For simplicity we denote $\phi^v_{\text{enc}}(\mathcal{V}^A)$ and $\phi^e_{\text{enc}}(\mathcal{E}^A)$ the updated node attributes and edge attributes by applying ϕ^v_{enc} and ϕ^e_{enc} to each node and each edge respectively. Then the encoder module can be briefly described as

$$\mathbb{G}^A \leftarrow \text{encode}(\mathbb{G}^A) = (\mathbb{V}^A, \mathbb{E}^A, \phi^v_{\text{enc}}(\mathcal{V}^A), \phi^e_{\text{enc}}(\mathcal{E}^A)). \tag{4}$$

The updated graph $\mathbb{G}^A$ is then passed to the subsequent convolution modules as input.

The decoder module reads out the final output from the updated graph state. Since only the node labels are required for final evaluation, the decoder module contains only one update function ϕ^v_{dec} that transforms the node attributes into a desired space

$$Y = \text{decode}(\mathbb{G}^A) = \phi^v_{\text{dec}}(\mathcal{V}^A) \tag{5}$$

where $Y \in \mathbb{R}^{n^2 \times 2}$ denotes the predicted node labels. Similarly, ϕ^v_{dec} is parameterized by an MLP and is mapped across all nodes to form per-node label.

4.3. The Convolution Module

The convolution module consists of an edge convolution layer and a node convolution layer, which aggregates the information of the neighborhoods of each node to update its attributes. The edge convolution layer assembles the attributes of the two nodes associated with each edge to generate a new attribute of this edge. It is followed by the node convolution layer that collects the attributes of all the edges adjacent to each node to compute per-node updates. Note that each instance of the convolution module aggregates the information of the 1^{st}-order neighborhoods, and thus the stack of k instances can convolve the k^{th}-order neighborhoods for each node. These neighborhoods serve as the receptive fields of a convolutional architecture, allowing our framework to learn effective graph representations.

The edge convolution layer: This layer is to update the edge attributes in $\mathbb{G}^A$ by aggregating the information of associated nodes for each edge. It consists of an aggregation function ρ^e and an update function ϕ^e

$$\begin{aligned} \bar{\mathbf{e}}_i &= \rho^e(\mathbf{v}_{s_i}, \mathbf{v}_{r_i}), \\ \mathbf{e}'_i &= \phi^e(\bar{\mathbf{e}}_i, \mathbf{e}_i), \end{aligned} \tag{6}$$

where $\mathbf{e}_i$ denotes the attributes of the i-th edge in $\mathbb{G}^A$, $\mathbf{v}_{s_i}$ and $\mathbf{v}_{r_i}$ the attributes of its source node and receive node respectively. The edge convolution layer uses computed $\mathbf{e}'_i$ to update the edge attribute and takes the updated assignment graph as output. Specifically, the aggregate function ρ^e is parameterized as

$$\rho^e(\mathbf{v}_{s_i}, \mathbf{v}_{r_i}) = M(\mathbf{v}_{s_i} \odot \mathbf{v}_{r_i}), \tag{7}$$

where $M \in \mathbb{R}^{d_e \times d_v}$ is the parameter matrices and $\odot$ denotes element-wise multiplication of two vectors. The update function ϕ^e is specified as an MLP that takes the concatenated vector $[\bar{\mathbf{e}}_i; \mathbf{e}_i]$ as input and outputs an updated edge attribute vector. Similar to cases in the encoder, ρ^e and ϕ^e are shared across all edges to compute per-edge affinity.

The node convolution layer: For each node, this layer aims to collect attributes of adjacent edges for node convolution. The computation of node convolution is performed by an aggregation function ρ^v and an update function ϕ^v

$$\begin{aligned} \bar{\mathbf{v}}_i &= \rho^v(\mathbb{E}_i^A), \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{v}}_i, \mathbf{v}_i), \end{aligned} \tag{8}$$

where $\mathbb{E}_i^A$ denote the set of all edge associated with the i-th node in $\mathbb{G}^A$. Different from the aggregating function ρ^e in the edge convolution layer, ρ^v in this layer is nonparametric and formulated as

$$\rho^v(\mathbb{E}_i^A) = \sum_{k \in \mathbb{E}_i^A} \mathbf{e}_k. \tag{9}$$

Similar to ϕ^e in the edge convolution layer, the update function ϕ^v is parameterized by an MLP that takes the concatenated vector $[\bar{\mathbf{v}}_i; \mathbf{v}_i]$ as input.

In summary, each instance of the convolution module contains one nonparametric function, ρ^v, and three trainable functions, ρ^e, ϕ^e and ϕ^v. In particular, these functions in different instances stacked in our framework are independent to each other, which allows our framework to aggregate information of neighborhoods at different orders with different degrees. It is notable that these functions are convolutional filters, *i.e.*, they are translation invariant to spatial locations of nodes and edges. That is, they perform per-node and per-edge computations that are independent of the input data structure, allowing our framework to perform end-to-end training for graphs with variable size and structure.

The number of Convolutional layers depends largely on the size of the constructed assignment graph, and we empirically fix it to 10 throughout our experiments.

4.4. Loss function

Similar to [27], we directly utilize the ground-truth node-to-node correspondence to guide the training of our network. Taking the ground-truth permutation matrix X^{gt} between $\mathbb{G}^{(1)}$ and $\mathbb{G}^{(2)}$, we first convert it to node labels $Y^{gt} \in \{0,1\}^{n^2 \times 2}$ of the assignment graph $\mathbb{G}^A$ as

$$\begin{cases} Y_{i,1}^{gt} = 1 - \mathbf{x}_i^{gt} \\ Y_{i,2}^{gt} = \mathbf{x}_i^{gt} \end{cases}, \text{ for } 1 \le i \le n^2, \tag{10}$$

where $\mathbf{x}^{gt} \doteq \text{vec}(X^{gt})$ is the vectorized version of the ground-truth permutation matrix. To evaluate the difference between the predicted node labels Y and the ground-truth node labels Y^{gt}, we adopt *cross entropy loss* to compute a permutation loss

$$\mathcal{L}^{perm} = -\sum_{i=1}^{n^2} (Y_{i,1}^{gt} \log Y_{i,1} + Y_{i,2}^{gt} \log Y_{i,2}). \tag{11}$$

The one-to-one matching constraints of graph matching usually play crucial roles in guiding graph matching in traditional non-learning algorithms [20, 29, 38], but it has not been fully explored in previous deep learning algorithms [27, 33]. In particular, these constraints can be formulated as

$$B\mathbf{x}^{gt} = \mathbf{1}_{2n}, \tag{12}$$

where $B \in \{0,1\}^{2n \times n^2}$ is an auxiliary matrix (please refer to [29] for the details of the construction of B). To impose the one-to-one matching constraints in our graph network, we first compute an index vector $\mathbf{y} \in \{0,1\}^{n^2}$ such that

$$\mathbf{y}_i = \arg\max_{k \in \{0,1\}} Y_{i,k+1}, \text{ for } 1 \le i \le n^2, \tag{13}$$

and then evaluate the degree of satisfaction of the one-to-one matching constraints by a constraint loss

$$\mathcal{L}^{con} = \|B(\mathbf{y} - \mathbf{x}^{gt})\|_2. \tag{14}$$

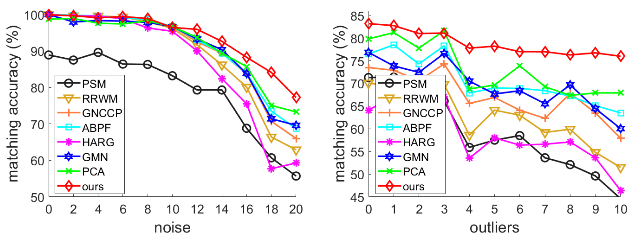
We finally combine the permutation loss and the constraint loss

$$\mathcal{L} = \mathcal{L}^{perm} + \alpha \mathcal{L}^{con}, \tag{15}$$

as the supervised information for end-to-end training of our graph network, where $\alpha \ge 0$ controls the degree of the one-to-one constraints imposed. In practice we set $\alpha = 0$ at the beginning of training and increase it gradually as training goes on.

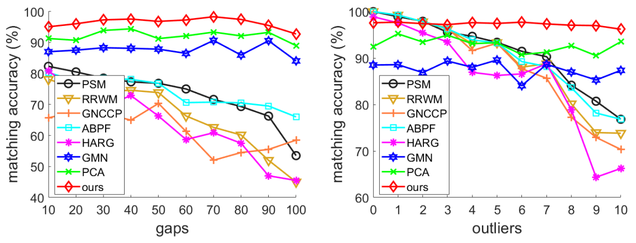
5. Experiments

We evaluate the proposed algorithm on four benchmarks in comparison with eight state-of-the-art algorithms, IPFP [15], RRWM [6], PSM [10], GNCCP [20], ABPF [29], HARG [5], GMN [33] and PCA [27], of which the last two are deep learning methods.



(a) varying noise (b) varying outliers

Figure 3. Comparison on 2D point sets.



(a) varying sampling gap varying outlier

Figure 4. Comparison on the CMU house dataset.

5.1. Synthetic 2D Points

We firstly compare these algorithms on the task of finding correspondences between sets of 2D points generated following the experimental protocol in [13]. For each trial, we construct two sets, $\mathbb{P}^{(1)}$ and $\mathbb{P}^{(2)}$, with $n_{\text{in}} = 10$ inlier points, and later add n_{out} outlier points to both sets. The inlier points in $\mathbb{P}^{(1)}$ are randomly selected in a given region of the plane. We then obtain the corresponding inliers in $\mathbb{P}^{(2)}$ by disturbing independently the points from $\mathbb{P}^{(1)}$ with white gaussian noise $\mathcal{N}(0, \sigma)$, and then rotating and translating the whole data set $\mathbb{P}^{(2)}$ with a random rotation and translation. The outlier points of each set are randomly selected from the same region as liners. The range of the point coordinates in $\mathbb{P}^{(1)}$ is $256\sqrt{(n_{\text{in}} + n_{\text{out}})/10}$. We generate 200,000 random pairs of point sets for training, in which n_{out} varies from 0 to 10 and σ from 0 to 20. Given two pairs of points $(i, j) \in \mathbb{P}^{(1)}$ and $(a, b) \in \mathbb{P}^{(2)}$, The edge affinity used in the non-learning methods is computed following [13].

We compare the performance of the algorithms under two different settings: (1) we increase noise parameter σ from 0 to 20 while fixing $n_{\text{out}} = 5$, and (2) we change the number of outliers n_{out} from 0 to 10 and set $\sigma = 20$. Under each scenario we generate 500 random pairs of graphs for testing. As illustrated in Fig. 3, our algorithm exhibits stronger robustness against both noise and outlier than compared baseline algorithms.

Table 1. Comparison of matching accuracy (%) on the Willow dataset.

Algorithm	Car	Duck	Face	Motor.	Wine.	AVG
IPFP [15]	74.8	60.6	98.9	84.0	79.0	79.5
RRWM [6]	86.3	75.5	**100**	94.9	94.3	90.2
PSM [10]	88.0	76.8	**100**	96.4	97.0	91.6
GNCCP [20]	86.4	77.4	**100**	95.6	95.7	91.0
ABPF [29]	88.4	80.1	**100**	96.2	96.7	92.3
HARG [5]	71.9	72.2	93.9	71.4	86.1	79.1
GMN [33]	74.3	82.8	99.3	71.4	76.7	80.9
PCA [27]	84.0	**93.5**	**100**	76.7	96.9	90.2
ours	**91.2**	86.2	**100**	**99.4**	**97.9**	**94.9**

5.2. CMU House Dataset

The CMU house dataset is widely-used for graph matching, which includes 111 frames of image sequences, where all sequences contain the same house object with transformation cross sequence gaps. In order to assess the matching accuracy, 30 landmarks were manually tracked and labeled across all frames.

We generate 300,000 random graph pairs for training. For each trial at training, we form graph pairs by randomly choosing two examples out of the 111 frames. To evaluate graph matching algorithms against noise, $n_1 (10 \leq n_1 \leq 30)$ points are randomly chosen out of the 30 landmark points for the first example of the pair, which implies the second example contains $30 - n_1$ outlier points.

We follow [29, 38] to generate the affinity matrix that is used in the non-learning algorithms. Graph edges are built by Delaunay triangulation [7], and each edge (i, j) is associated with a weight w_{ij} that is computed as the Euclidean distance between the connected nodes v_i and v_j. The node affinity is set to zero, and the edge affinity between edges (i, j) in $\mathbb{G}^{(1)}$ and (a, b) in $\mathbb{G}^{(2)}$ is computed as $K_{\text{ind}(i,a)\text{ind}(j,b)} = \exp(-(w^{(1)}_{ij} - w^{(2)}_{ab})^2/2500)$.

The matching accuracy of the compared methods are tested under two scenarios. In the first case (Fig. 4(a)) we set $n_1 = 20$ and match all possible image pairs, in total 560 pairs gapped by 10, 20, ..., 100 frames, where increasing sampling gaps implies the increase of deformation degree. In the second (Fig. 4(b)) we fix the sequence gap to 50 and vary n_1 from 30 to 20, i.e., increasing the number of outliers from 0 to 10. It is observed that the non-learning methods gain perfect matching results when no outlier point exists, but increasing outliers degrades greatly their performance. In contrast, the deep learning methods (GMN [33], PCA [27] and ours) are relatively robust against deformation and outlier by learning a more robust affinity function. In particular, our method outperforms all baseline methods in general, and achieves about 95% matching accuracy under all tested scenarios.

Table 2. Comparison of matching accuracy (%) on the Pascal VOC dataset.

Algorithm	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	AVG
IPFP [15]	25.1	26.4	41.4	50.3	43.0	32.9	37.3	32.5	33.6	28.2	26.9	26.1	29.9	32.0	28.8	62.9	28.2	45.0	69.3	33.8	36.6
RRWM [6]	30.9	40.0	46.4	54.1	52.3	35.6	47.4	37.3	36.3	34.1	28.8	35.0	39.1	36.2	39.5	67.8	38.6	49.4	70.5	41.3	43.0
PSM [10]	32.6	37.5	49.9	53.2	47.8	34.6	50.1	35.5	37.2	36.3	23.1	32.7	42.4	37.1	38.5	62.3	41.7	54.3	72.6	40.8	43.1
GNCCP [20]	28.9	37.1	46.2	53.1	48.0	36.3	45.5	34.7	36.3	34.2	25.2	35.3	39.8	39.6	40.7	61.9	37.4	50.5	67.0	34.8	41.6
ABPF [29]	30.9	40.4	47.3	54.5	50.8	35.1	46.7	36.3	40.9	38.9	16.3	34.8	39.8	39.6	39.3	63.2	37.9	50.2	70.5	41.3	42.7
GMN [33]	31.9	47.2	51.9	40.8	68.7	72.2	53.6	52.8	34.6	48.6	72.3	47.7	54.8	51.0	38.6	75.1	49.5	45.0	83.0	86.3	55.3
PCA [27]	40.9	55.0	**65.8**	47.9	76.9	77.9	63.5	**67.4**	33.7	**65.5**	63.6	**61.3**	**68.9**	**62.8**	44.9	77.5	**67.4**	57.5	86.7	90.9	63.8
ours	**46.9**	**58.0**	63.6	**69.9**	**87.8**	**79.8**	**71.8**	60.3	**44.8**	64.3	**79.4**	57.5	64.4	57.6	**52.4**	**96.1**	62.9	**65.8**	**94.4**	**92.0**	**68.5**

5.3. Willow Object Dataset

The Willow Object dataset is provided by [5] containing images of five classes, namely car, duck, face, motorbike and winebottle from Caltech-256 and Pascal VOC 2007. Each class contains at least 40 images with different instances and 10 distinctive landmarks were manually labeled on the target object across all images in each class.

Following [5] we randomly choose 20 images from each object class for training and keep the rest for testing. For testing we randomly select 1000 pairs of images from the testing set of each class respectively.

To generate the affinity matrix used in the non-learning algorithms, we follow [29] using SIFT descriptor [21] to represent the node attributes and compute the node affinity via their appearance similarity. Delaunay triangulation is adopted to build graph edges, and each edge (i, j) is associated with two features d_{ij} and θ_{ij}, where d_{ij} is the pairwise distance between the connected nodes v_i and v_j, and θ_{ij} is the absolute angle between the edge and the horizontal line, i.e., $0 \leq \theta_{ij} \leq \pi/2$. Consequently, the edge affinity between edges (i, j) in $\mathbb{G}^{(1)}$ and (a, b) in $\mathbb{G}^{(2)}$ is computed as $K_{\text{ind}(i,a)\text{ind}(j,b)} = \exp(-(|d_{ij} - d_{ab}| + |\theta_{ij} - \theta_{ab}|)/2)$.

Table 1 shows the matching accuracy of our algorithm in comparison with baseline algorithms[2]. This dataset is considered relatively easy because the lack of pose, scale and illumination changes. The predefined affinity function is sufficient to capture the structure consistency, and thus the previous deep learning methods [27, 33] that train only the affinity function illustrate less superiority to the non-learning methods. In contrast, our method is able to learn not only the affinity function but also the combinatorial solver for graph matching. As a result, our method achieves the best matching results in all object classes except the Duck class, and rises the average matching accuracy up to 94.9%.

5.4. Pascal VOC Keypoints

This dataset is an extension of the PASCAL VOC dataset with Berkeley annotations of keypoints [3] that contains 20 classes of instances with labeled keypoint locations. This dataset is considered more difficult than the Willow dataset because instances may vary from its scale, pose and illumination, and the number of inliers ranges from 6 to 23.

For fair evaluation we use the same filtered dataset as that in [27], which contains 7,020 annotated images for training and 1,682 for testing. As a training sample can be formed using any two images of the same class, we draw randomly 100,000 samples from each class for training.

The method of affinity generation for the non-learning algorithms is the same as that in Sec. 5.3. As illustrated in Table 2, our algorithm outperforms not only non-learning baselines but also all learning-based baselines. Note that non-learning algorithms show remarkable inferiority to the learning-based algorithms on this complex dataset for that the handcrafted affinity function is unable to capture the structure consistency due to the noise and outliers.

6. Conclusion

In this paper, we proposed a novel deep learning algorithm for graph matching aiming to improve the matching accuracy. We first convert the problem of building node correspondences between input graphs to the problem of selecting reliable nodes from a constructed assignment graph. In order to solve for node classification, we propose a fully trainable network that embeds the graph network block module to form structured representations for each node by convolving its neighborhoods. Moreover, a new loss function that encodes the one-to-one matching constraints is proposed to guide the training of our network. Experimental results reveal that our graph matching algorithm gains robustness against noises and outliers, and outperforms state-of-the-art algorithms.

Acknowledgment. This work is supported by the Fundamental Research Funds for the Central universities (No. 2018JBM015), the China National Key Research and Development Plan (No. 2016YFB1001200), the National Nature Science Foundation of China (Nos. 61673048 and 61872032), and the Beijing Municipal Natural Science Foundation (No.4202060). The work was supported partially by the Yahoo Faculty Research and Engagement Program Award and the Amazon AWS Machine Learning Research Award. The corresponding authors are Yidong Li and Haibin Ling.

[2]The results of [5, 27, 33] are cited from [27].

References

[1] Kamil Adamczewski, Yumin Suh, and Kyoung Mu Lee. Discrete tabu search for graph matching. In *ICCV*, pages 109–117, 2015.

[2] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[3] Lubomir D. Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, pages 1365–1372, 2009.

[4] Tibério S. Caetano, Julian J. McAuley, Li Cheng, Quoc V. Le, and Alexander J. Smola. Learning graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(6):1048–1058, 2009.

[5] Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *CVPR*, pages 25–32, 2013.

[6] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *ECCV*, pages 492–505, 2010.

[7] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Dewall: A fast divide and conquer delaunay triangulation algorithm in Ed. *Computer-Aided Design*, 30(5):333–341, 1998.

[8] Timothée Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In *NIPS*, pages 313–320, 2006.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3837–3845, 2016.

[10] Amir Egozi, Yosi Keller, and Hugo Guterman. A probabilistic approach to spectral graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):18–27, 2013.

[11] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(4):377–388, 1996.

[12] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[13] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, pages 1482–1489, 2005.

[14] Marius Leordeanu and Martial Hebert. Smoothing-based optimization. In *CVPR*, 2008.

[15] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and MAP inference. In *NIPS*, pages 1114–1122, 2009.

[16] Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International Journal of Computer Vision*, 96(1):28–45, 2012.

[17] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *ICCV*, pages 2274–2281, 2011.

[18] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

[19] C. Karen Liu, Aaron Hertzmann, and Zoran Popovic. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, 2005.

[20] Zhiyong Liu and Hong Qiao. GNCCP - graduated nonconvexityand concavity procedure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(6):1258–1267, 2014.

[21] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[22] Gengyu Lyu, Songhe Feng, Tao Wang, Congyan Lang, and Yidong Li. GM-PLL: graph matching based partial label learning. *IEEE Trans. Knowl. Data Eng.*, 2019.

[23] Eduard Serradell, Miguel Amvel Pinheiro, Raphael Sznitman, Jan Kybic, Francesc Moreno-Noguer, and Pascal Fua. Non-rigid graph registration using active testing search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(3):625–638, 2015.

[24] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *ECCV*, pages 596–609, 2008.

[25] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005.

[26] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

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

[28] Tao Wang and Haibin Ling. Gracker: A graph-based planar object tracker. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(6):1494–1501, 2018.

[29] Tao Wang, Haibin Ling, Congyan Lang, and Songhe Feng. Graph matching with adaptive and branching path following. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(12):2853–2867, 2018.

[30] Tao Wang, Haibin Ling, Congyan Lang, Songhe Feng, and Xiaohui Hou. Deformable surface tracking by graph matching. In *ICCV*, pages 901–910, 2019.

[31] Jun Wu, Hong Shen, YiDong Li, ZhiBo Xiao, MingYu Lu, and ChunLi Wang. Learning a hybrid similarity measure for image retrieval. *Pattern Recognition*, 46(11):2927–2939, 2013.

[32] Junchi Yan, Chao Zhang, Hongyuan Zha, Wei Liu, Xiaokang Yang, and Stephen M. Chu. Discrete hyper-graph matching. In *CVPR*, pages 1520–1528, 2015.

[33] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *CVPR*, pages 2684–2693, 2018.

[34] Mikhail Zaslavskiy, Francis R. Bach, and Jean-Philippe Vert. A path following algorithm for the graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2227–2242, 2009.

[35] Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *CVPR*, 2008.

[36] Quanshi Zhang, Xuan Song, Xiaowei Shao, Huijing Zhao, and Ryosuke Shibasaki. Learning graph matching: Oriented to category modeling from cluttered scenes. In *ICCV*, pages 1329–1336, 2013.

[37] Quanshi Zhang, Xuan Song, Xiaowei Shao, Huijing Zhao, and Ryosuke Shibasaki. Attributed graph mining and matching: An attempt to define and extract soft attributed patterns. In *CVPR*, pages 1394–1401, 2014.

[38] Feng Zhou and Fernando De la Torre. Factorized graph matching. In *CVPR*, pages 127–134, 2012.