Hypergraph Neural Networks for Hypergraph Matching

Xiaowei Liao[1,2] Yong Xu[1,2,3,*] Haibin Ling[4,*]
[1]School of Computer Science & Engineering, South China Univ. of Tech., Guangzhou, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Communication and Computer Network Laboratory of Guangdong, China
[4]Department of Computer Science, Stony Brook University, Stony Brook, NY USA
liao.xw@mail.scut.edu.cn, yxu@scut.edu.cn, hling@cs.stonybrook.edu

Abstract

Hypergraph matching is a useful tool to find feature correspondence by considering higher-order structural information. Recently, the employment of deep learning has made great progress in the matching of graphs, suggesting its potential for hypergraphs. Hence, in this paper, we present the first, to our best knowledge, unified hypergraph neural network (HNN) solution for hypergraph matching. Specifically, given two hypergraphs to be matched, we first construct an association hypergraph over them and convert the hypergraph matching problem into a node classification problem on the association hypergraph. Then, we design a novel hypergraph neural network to effectively solve the node classification problem. Being end-to-end trainable, our proposed method, named HNN-HM*, jointly learns all its components with improved optimization. For evaluation, HNN-HM is tested on various benchmarks and shows a clear advantage over state-of-the-arts.*

1. Introduction

Feature correspondence is essential for many computer vision tasks, such as shape matching [2], image registration [17], and object recognition [26]. Given two sets of features, feature correspondence aims to match each feature in one set to a feature in the other set. Usually, features in the same set are related and have an inherent structure, which can benefit the corresponding task. Using the pairwise relations between features (second-order structure), one can build two graphs from the two feature sets (nodes represent features, and edges represent relations) and convert the corresponding problem into a graph matching problem [14,33]. Similarly, to incorporate higher-order structural information (*i.e*. relations involving more than two features), one can build two hypergraphs and convert the problem into a hypergraph matching problem [10,44].

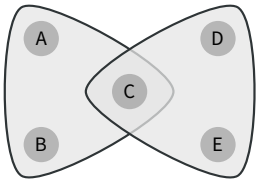


Figure 1. A hypergraph. This hypergraph has five nodes (A, B, C, D, and E) and two hyperedges ($\{A, B, C\}$ and $\{C, D, E\}$).

A hypergraph (illustrated in Fig. 1) is a generalization of a graph, stimulated by the idea that each hyperedge captures the relation among multiple (usually more than two) nodes. The task of hypergraph matching is to find the node correspondence between two given hypergraphs by considering the affinities of their corresponding nodes and hyperedges. Given the affinities of nodes and hyperedges, a hypergraph matching problem can be formulated as a challenging combinatorial optimization problem. Most hypergraph matching algorithms focus on how to approximately solve that optimization problem [10, 19, 20, 27, 28, 40, 44]. While these algorithms have kept pushing the frontier of performance, they are practically limited due to the hand-crafted affinities and task-agnostic combinatorial solver. Recently, the employment of deep learning has made great progress in the research of graph matching problems [12,30,36,39,41,42], suggesting its potential for hypergraph matching problems.

Inspired by the work mentioned above, in this paper, we present the first, to our best knowledge, unified hypergraph neural network (HNN) solution for hypergraph matching (illustrated in Fig. 2). Specifically, given two hypergraphs $\mathcal{G}^1$ and $\mathcal{G}^2$ to be matched, we first construct an association hypergraph $\mathcal{G}^a$ over them. Each node in $\mathcal{G}^a$ represents a pair of nodes from $\mathcal{G}^1$ and $\mathcal{G}^2$, and each hyperedge in $\mathcal{G}^a$ captures the higher-order relationship between two hyperedges from $\mathcal{G}^1$ and $\mathcal{G}^2$. With $\mathcal{G}^a$, the matching between $\mathcal{G}^1$ and $\mathcal{G}^2$ is converted to a node selection problem on $\mathcal{G}^a$.

Then, we propose a novel hypergraph neural network to

*Corresponding authors.

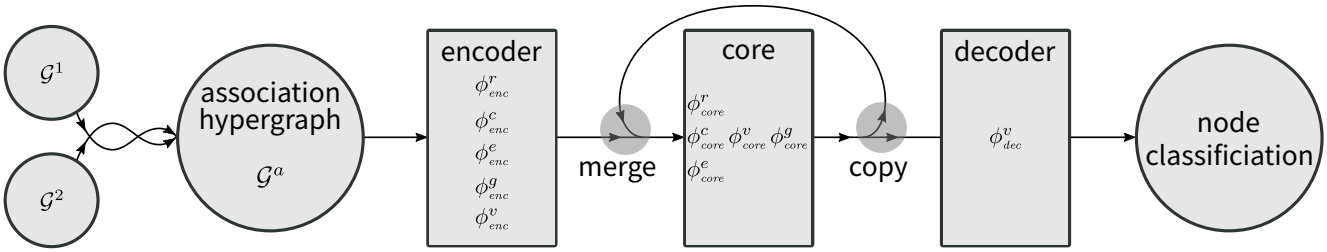


Figure 2. An illustration of our proposed hypergraph neural network (HNN-HM). To match hypergraphs $\mathcal{G}^1$ and $\mathcal{G}^2$, we first construct an association hypergraph $\mathcal{G}^a$ over them and convert the matching problem into a node classification problem on $\mathcal{G}^a$. The classification problem is then solved by our proposed HNN-HM, which comprises several hypergraph neural network blocks (the encoder module, the core module, and the decoder module). The encoder module transforms $\mathcal{G}^a$ from the input space to an embedding space, using update functions (ϕ^r_{enc}, ϕ^c_{enc}, ϕ^e_{enc}, ϕ^v_{enc}, and ϕ^g_{enc}) for each attribute independently. Then the core module is applied multiple times to update the state of the $\mathcal{G}^a$ in the embedding space, in which the update functions for hyperedges (ϕ^r_{core}, ϕ^c_{core}, ϕ^e_{core}), the update function for nodes (ϕ^v_{core}), and the update function for the global attribute (ϕ^g_{core}) are applied sequentially. Finally, the decoder module transforms the association hypergraph from the embedding space to the desired probability space, using an update function for nodes (ϕ^v_{dec}).

solve the node selection/classification problem. The proposed network is comprised of several hypergraph neural network blocks. Each block accepts an association hypergraph as input and returns an updated version as output. Concretely, in each block, node attributes related to a hyperedge will be aggregated to update the hyperedge attributes; and the updated hyperedge attributes will also be aggregated to update the attributes of related nodes. This way, the structure of the association hypergraph will be utilized to update its attributes dynamically. Both the node and hyperedge update operations are shift-invariant, thus waiving the limitation of the size of the association hypergraph. Finally, we can decode the updated state of the association hypergraph, and reinterpret the attribute of each node as the probability that the node will be selected.

The proposed method, denoted by HNN-HM, is end-to-end trainable and hence allows all components to be jointly optimized. In particular, it allows affinity learning and combinatorial optimization to be learned together to assimilate their interaction. For evaluation, HNN-HM is tested on four benchmarks in comparison with seven state-of-the-art hypergraph matching algorithms. The results clearly show the advantage of our method over previous solutions.

In summary, our main contributions include

- we present the first unified hypergraph neural network (HNN) solution for hypergraph matching;
- we convert the problem of hypergraph matching into a node classification problem and develop a hypergraph neural network to solve it; and
- we test our proposed HNN-HM on various benchmarks and achieve state-of-the-art results.

The source code of our work is made available at `https://github.com/xwliao/HNN-HM`.

2. Related work

2.1. Graph matching

Algorithms of graph matching can be roughly divided into two categories: non-learning-based (*e.g.*, [7–9, 16, 21, 22, 25, 38, 43, 45]) and learning-based (*e.g.*, [5, 6, 12, 23, 30, 36, 39, 41, 42]).

Non-learning-based algorithms typically treat graph matching as a quadratic assignment problem (QAP), which is known to be NP-hard, and pursue an approximate solution often by relaxing either the loss function or the constraints. Among them, SM [21] relaxes the permutation matrix constraint to matrices with unit Frobenius norm, hence solving the simplified problem by finding the principal eigenvector of the affinity matrix. Based on that, further consideration of the affine constraint leads to SMAC [9]. To obey the discrete constraints, IPFP [22] injects linear assignment solver and line search method into the power iteration algorithm. RRWM [7] transforms the graph matching problem into a random walk problem on an association graph and introduces constraints in its iterative subprocess. PATH [43] relaxes the graph matching problem to a convex optimization problem and a concave optimization problem (which has the same optimal solution) and uses a path-following strategy to approximately solve a sequence of weighted combined problems. BPF [38] improves the path following strategy by detecting singular points and switching to a better solution path. FGM [45] decomposes the affinity matrix in the graph matching problem into smaller components and follows the path-following strategy to solve it iteratively. GNCCP [25] follows a convex-concave relaxation procedure without explicitly expressing the relaxations.

Learning-based algorithms parametrize some or all components of the graph matching problem and learn these components in a data-driven way. Considering the construc-

tion of the input graphs, [6] suggests learning task-driven graphs for different graph matching tasks. GMN [42] designs the first end-to-end deep learning framework to solve graph matching problems. PCA-GM [36] adopts the graph convolutional network [18] to embed the graph structure into node features. However, both deep learning algorithms learn only graph attributes and use a fixed Sinkhorn algorithm to find the feature correspondence solution. To learn also the combinatorial solver, [39] transforms the graph matching problem into a node classification problem on an association graph and adopts a graph neural network to solve the transformed problem. Although it has achieved significant success, it is limited to second-order graphs. Our work is inspired by [39] but breaks its limitation of second-order graphs and performs comparably to or better than it. DGMC [12] uses graph neural networks (GNNs) and the Sinkhorn algorithm to estimate and refine node correspondence iteratively, and with the help of SplineCNN [13], it performs quite well. Instead of learning a combinatorial solver, BB-GM [30] embeds an existing well-designed combinatorial solver and learns appropriate feature extractors in an end-to-end way by computing differentiable interpolation of the combinatorial solver based on their previous work [35]. By using SplineCNN [13] to extract node embedding, BB-GM [30] obtains very impressive results.

2.2. Hypergraph matching

Compared with graph matching, the research on hypergraph matching is relatively new [10, 19, 20, 24, 27, 28, 37, 40, 44], and many of these methods are direct extensions of previous graph matching algorithms.

TM [10], which is an extension of SM [21], finds an approximate solution of the hypergraph matching problem by estimating the rank-1 approximation of the affinity tensor. The work in [24] introduces learning to hypergraph matching and presents a hypergraph matching algorithm that performs sequential second-order approximation (based on IPFP [22]). RRWHM [20] transforms the hypergraph matching problem into a random walk problem on an association hypergraph and solves it in a similar way to RRWM [7]. From the perspective of probability, and assuming that the matchings between nodes are independent after the structure of the two hypergraphs to be matched are known, HGM [44] simplifies the problem as a linear assignment problem that can be solved exactly and efficiently. HADGA [40] approximates the hypergraph matching problem as sequential linear assignment problems using previous solutions. Although it can avoid the post discretization step and has the promise of convergence, it is bothered about the suboptimal approximation. Specific to the matching problem between two 3-uniform hypergraphs, BCAGM [27] promotes the third-order problem to a fourth-order multilinear problem with the same solution and then solves that problem by dealing with iterative first-order or second-order subproblems. Its successor [28] eliminates the need for order promotion and can also guarantee convergence. ADGM [19] rewrites the graph matching problem to an equivalent one with several decomposed variables respecting different constraints and solves it based on the ADMM [4] algorithm. In [31], a multi-hypergraph matching solution was proposed based on the rank-1 tensor approximation originally used for multi-target tracking [32]. Recently, [37] extended the PCA-GM [36] method to hypergraph and multiple-graph matching.

3. Hypergraph matching problem

This section presents the formalization of hypergraphs and hypergraph matching problems.

3.1. Hypergraph

For an attributed hypergraph[1] with N^v nodes and N^e hyperedges, we denote its i-th node and associated attribute vector by V_i and $\mathbf{v}_i$, and its j-th hyperedge and associated attribute vector by E_j and $\mathbf{e}_j$. Thus an attributed hypergraph can be indicated by a 5-tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbb{V}, \mathbb{E}, \mathbf{g})$, where

- $\mathcal{V} = \{V_i\}_{1 \le i \le N^v}$ indicates the set of nodes;
- $\mathcal{E} = \{E_j\}_{1 \le j \le N^e}$ indicates the set of hyperedges;
- $\mathbb{V} = \{v_i\}_{1 \le i \le N^v}$ indicates the multiset of attribute vectors associated with each node;
- $\mathbb{E} = \{e_j\}_{1 \le j \le N^e}$ indicates the multiset of attribute vectors associated with each hyperedge; and
- $\mathbf{g}$ indicates a global attribute vector associated with the entire hypergraph.

In addition, there are some related concepts as follows

- **K-uniform hypergraph**: each hyperedge is associated with exactly K nodes;
- **undirected hypergraph**: each hyperedge is a subset of $\mathcal{V}$, *e.g.*, $E_1 = \{V_1, V_2, V_3\}$;
- **directed hypergraph**[2]: each hyperedge is a tuple containing elements from $\mathcal{V}$, *e.g.*, $E_1 = (V_1, V_2, V_3)$.

In this paper, we define a hyperedge (E_j) as a K-tuple whose elements are *subsets* of $\mathcal{V}$:

$$E_j = (E_j^{(1)}, E_j^{(2)}, \dots, E_j^{(K)}) \quad \forall\, 1 \le j \le N^e, \tag{1}$$

where

$$E_j^{(k)} \subseteq \mathcal{V} \quad \forall\, 1 \le k \le K. \tag{2}$$

Thus an undirected hyperedge is the special case when $K = 1$, and can be denoted by $(\{V_{i_1}, V_{i_2}, \dots, V_{i_n}\})$, or $\{V_{i_1}, V_{i_2}, \dots, V_{i_n}\}$ for simplicity. And we can denote a K-uniform directed hyperedge by $(\{V_{i_1}\}, \{V_{i_2}\}, \dots, \{V_{i_K}\})$, or $(V_{i_1}, V_{i_2}, \dots, V_{i_K})$ for simplicity. Therefore, this notation is flexible to express a variety of hyperedges.

[1] We use hypergraphs and attributed hypergraphs indiscriminately.

[2] The directed hypergraph defined here is different from [15].

3.2. Hypergraph matching problem

Given two K-uniform directed hypergraphs $\mathcal{G}^1 = (\mathcal{V}^1, \mathcal{E}^1, \mathbb{V}^1, \mathbb{E}^1, \mathbf{g}^1)$ and $\mathcal{G}^2 = (\mathcal{V}^2, \mathcal{E}^2, \mathbb{V}^2, \mathbb{E}^2, \mathbf{g}^2)$, we want to find the node correspondence between them by considering the node affinities ($c_{ij} = \theta^v(\mathbf{v}_i^1, \mathbf{v}_j^2)$) and hyperedge affinities ($d_{\mathcal{I}\mathcal{J}} = \theta^e(\mathbf{e}_{\mathcal{I}}^1, \mathbf{e}_{\mathcal{J}}^2)$). Without loss of generality, we assume that $\left|\mathcal{V}^1\right| \leq \left|\mathcal{V}^2\right|$.

The node correspondence can be represented by an assignment matrix $\mathbf{X} \in \{0,1\}^{|\mathcal{V}^1|\times|\mathcal{V}^2|}$ that is defined as

$$\mathbf{X}_{i,j} = \begin{cases} 1 & \text{if } V_i^1 \text{ matches } V_j^2, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

And all affinities can be represented by an affinity tensor[3]

$$\mathcal{A}_{i_1 j_1,\ldots,i_K j_K} = \begin{cases} c_{ij} & \text{if } i = i_k \wedge j = j_k \quad \forall k, \\ d_{\mathcal{I}\mathcal{J}} & \text{if } \exists\, E_{\mathcal{I}}^1 \in \mathcal{E}^1 \wedge E_{\mathcal{J}}^2 \in \mathcal{E}^2, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathcal{I}$ is an index corresponding to $(i_1, i_2, \ldots, i_K)$ (one to one), $E_{\mathcal{I}}^1 = (V_{i_1}^1, V_{i_2}^1, \ldots, V_{i_K}^1)$ is any possible K-tuple of nodes from $\mathcal{G}^1$, and so do $\mathcal{J}$ and $E_{\mathcal{J}}^2 = (V_{j_1}^2, V_{j_2}^2, \ldots, V_{j_K}^2)$.

Given the affinity tensor, the hypergraph matching problem can be formalized as an optimization problem:

$$\max_{\mathbf{X}\in\mathcal{P}} \sum_{\substack{i_1,\ldots,i_K,\\ j_1,\ldots,j_K}} \mathcal{A}_{i_1 j_1,\ldots,i_K j_K} \mathbf{X}_{i_1,j_1} \cdots \mathbf{X}_{i_K,j_K}, \tag{5}$$

where $\mathcal{P}$ is the space of (partial) permutation matrices corresponding to the one-to-(at most)-one constraint.

There are two main challenges for hypergraph matching solvers. One is the imperfect affinity tensor (usually handcrafted), and the other is the intrinsic hardness of the optimization problem. Most of the previous hypergraph matching algorithms focused on the second challenge to find an approximate solution. However, we deal with these two challenges by jointly learning the affinities and the combinatorial solver in a unified hypergraph neural network.

4. The proposed hypergraph neural networks

This section presents a general hypergraph neural network (HNN-HM) framework. Details of how to apply it in hypergraph matching problems are explained in Section 5.

4.1. Hypergraph neural network block

The proposed hypergraph neural networks are composed of a stack of hypergraph neural network blocks (HNN-HM blocks). Each HNN-HM block accepts an attributed hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbb{V}, \mathbb{E}, \mathbf{g})$ as input and returns an updated attributed hypergraph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}, \mathbb{V}', \mathbb{E}', \mathbf{g}')$ as output. It use the hypergraph structure $(\mathcal{V}, \mathcal{E})$ to update the attributes $(\mathbb{V}, \mathbb{E}, \mathbf{g})$ and keep the hypergraph structure untainted.

Suppose that every hyperedge is a K-tuple of subsets (defined in Eq. (1)), we can define the update functions of a HNN-HM block as

$$\mathbf{e}_j' = \Phi^e(\mathbf{e}_j, \mathbb{V}_j^{(1)}, \mathbb{V}_j^{(2)}, \ldots, \mathbb{V}_j^{(K)}, \{\mathbf{g}\}), \tag{6a}$$

$$\mathbf{v}_i' = \Phi^v(\mathbf{v}_i, \mathbb{E}_i^{(1)'}, \mathbb{E}_i^{(2)'}, \ldots, \mathbb{E}_i^{(K)'}, \{\mathbf{g}\}), \tag{6b}$$

$$\mathbf{g}' = \Phi^g(\mathbf{g}, \mathbb{V}', \mathbb{E}'), \tag{6c}$$

where ($\forall\, 1 \leq k \leq K$)

$$\mathbb{V}_j^{(k)} = \left\{\mathbf{v}_i \mid V_i \in E_j^{(k)} \quad \forall\, 1 \leq i \leq N^v\right\}, \tag{7}$$

$$\mathbb{E}_i^{(k)'} = \left\{\mathbf{e}_j' \mid V_i \in E_j^{(k)} \quad \forall\, 1 \leq j \leq N^e\right\}, \tag{8}$$

$$\mathbb{V}' = \left\{\mathbf{v}_i' \mid \forall\, 1 \leq i \leq N^v\right\}, \tag{9}$$

$$\mathbb{E}' = \left\{\mathbf{e}_j' \mid \forall\, 1 \leq j \leq N^e\right\}. \tag{10}$$

Eq. (7) states that for the k-th node set $E_j^{(k)}$ in a hyperedge E_j, we collect all associated node attributes into a multiset $\mathbb{V}_j^{(k)}$. Similarly, Eq. (8) states that for each node V_i, we first find all hyperedges that contain this node in its k-th node set and then collect all associated hyperedge attributes (which have been updated in Eq. (6a)) into a multiset $\mathbb{E}_i^{(k)'}$. Thus the update function Eq. (6a) aggregates all node attributes that are related to a hyperedge, then uses them (combined with the global attribute) to update the attribute of that hyperedge. After that, by using Eq. (6b), each node will update its attribute by aggregating all related hyperedge attributes and the global attribute. Finally, using Eq. (6c), the global attribute will be updated by aggregating all updated attributes of nodes and hyperedges.

Note that all update functions in Eq. (6) share the form

$$\mathbf{y}' = \Phi(\mathbf{y}, \mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_N) \tag{11}$$

where each $\mathbb{X}_t$ is a multiset containing vectors with the same dimension. By introducing reduction functions

$$\mathbf{x}_t = \Psi_t(\mathbf{y}, \mathbb{X}_t) \quad \forall\, 1 \leq t \leq N, \tag{12}$$

we can further restrict Eq. (11) to have the form

$$\mathbf{y}' = \phi(\mathbf{y}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N), \tag{13}$$

where ϕ is usually a learnable neural network. Some useful reduction functions for $\Psi(\mathbf{y}, \mathbb{X})$ (ignore the subscript t) are

$$\Psi_{sum}(\mathbf{y}, \mathbb{X}) = \psi_{sum}(\mathbb{X}) = \sum_{\mathbf{x}\in\mathbb{X}} \mathbf{x}; \tag{14}$$

$$\Psi_{mean}(\mathbf{y}, \mathbb{X}) = \psi_{mean}(\mathbb{X}) = \frac{1}{|\mathbb{X}|} \sum_{\mathbf{x}\in\mathbb{X}} \mathbf{x}; \tag{15}$$

$$\Psi_{atten}(\mathbf{y}, \mathbb{X}) = \sum_{\mathbf{x}\in\mathbb{X}} attention(\mathbf{y}, \mathbf{x}) \cdot \mathbf{x}. \tag{16}$$

[3]For simplicity, all nodes of any hyperedge are not exactly the same. Note that the affinity tensor is of size $(|\mathcal{V}^1| \cdot |\mathcal{V}^2|)^K$, but it is usually sparse.

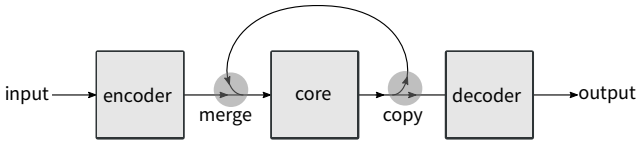


Figure 3. An encode-process-decode model with recurrence.

For a 2-uniform directed hypergraph (*i.e*. a directed graph), if we let $E_j = (\{j\text{-th tail node}\}, \{j\text{-th head node}\})$ denote each hyperedge and let $\Psi(\mathbf{y}, \mathbb{X}) = \psi(\mathbb{X})$, then our HNN-HM block degenerates to the GN block [1].

Since all hyperedges share the same update function Eq. (6a) and all nodes share the same Eq. (6b), our proposed HNN-HM block have the same benefit as the GN block [1]: it does not assume the specific structure of the hypergraph and can be applied to arbitrary K-fixed [4] hypergraphs. Moreover, by adding different kinds of hyperedges and let each kind owns its update function, the proposed HNN-HM block can incorporate more types of information.

4.2. The hypergraph neural networks

The HNN-HM block is the core of our HNN-HM. We can run a block multiple times or stack different blocks to control different types/levels of information aggregation. Following [1], we use an encode-process-decode model with recurrent structure (see Fig. 3). All components (the *encoder* module, the *core* module, and the *decoder* module) in the architecture are HNN-HM blocks. The encoder module transforms the input hypergraph from the input space to an embedding space suitable for later processing. After that, the core module is applied multiple times to update the state of the hypergraph in the embedding space. Finally, the decoder module transforms the hypergraph from the embedding space to the desired output space.

5. Hypergraph neural networks for hypergraph matching

There is still a gap between the hypergraph matching problem and the proposed HNN-HM: we have two hypergraphs to be matched while the proposed network accepts only one. One may use HNN-HM to update each hypergraph separately, then pass them to a combinatorial solver. However, this may suffer from the sub-optimality of the combinatorial solver. Instead, we convert the matching problem into a node classification problem on an association hypergraph and deliver it to our HNN-HM. In this section, we first explain the creation of association hypergraphs and then detail all components of our HNN-HM framework.

[4] It does require that K is the same for all input hypergraphs.

5.1. Association hypergraph

Consider a pair of K-uniform directed hyperedges (from two hypergraphs $\mathcal{G}^1$ and $\mathcal{G}^2$, separately):

$$\left(E^1_{\mathcal{I}}, E^2_{\mathcal{J}}\right) = \left(\left(V^1_{i_1}, \ldots, V^1_{i_K}\right), \left(V^2_{j_1}, \ldots, V^2_{j_K}\right)\right), \quad (17)$$

we can rearrange it to

$$E^a_{\mathcal{I}\mathcal{J}} = \left(\left(V^1_{i_1}, V^2_{j_1}\right), \left(V^1_{i_2}, V^2_{j_2}\right), \ldots, \left(V^1_{i_K}, V^2_{j_K}\right)\right) \quad (18)$$

The insight behind the concept of association hypergraph is that: we can regard $V^a_{ij} = \left(V^1_i, V^2_j\right)$ as a node in a new hypergraph and regard $E^a_{\mathcal{I}\mathcal{J}} = \left(V^a_{i_1 j_1}, V^a_{i_2 j_2}, \ldots, V^a_{i_K j_K}\right)$ as a hyperedge (with K nodes) in the new hypergraph. The created new hypergraph is an association hypergraph.

More specifically, the association hypergraph over two K-uniform directed hypergraphs ($\mathcal{G}^1$ and $\mathcal{G}^2$) can be defined as an K-uniform directed hypergraph $\mathcal{G}^a = (\mathcal{V}^a, \mathcal{E}^a, \mathbb{V}^a, \mathbb{E}^a, \mathbf{g}^a)$ with the following components:

$$\mathcal{V}^a = \left\{V^a_{ij} \mid \forall\, V^1_i \in \mathcal{V}^1 \wedge V^2_j \in \mathcal{V}^2\right\}, \quad (19)$$

$$\mathcal{E}^a = \left\{E^a_{\mathcal{I}\mathcal{J}} \mid \forall\, E^1_{\mathcal{I}} \in \mathcal{E}^1 \wedge E^2_{\mathcal{J}} \in \mathcal{E}^2\right\}, \quad (20)$$

$$\mathbb{V}^a = \left\{\theta^v(\mathbf{v}^1_i, \mathbf{v}^2_j) \mid \forall\, V^1_i \in \mathcal{V}^1 \wedge V^2_j \in \mathcal{V}^2\right\}, \quad (21)$$

$$\mathbb{E}^a = \left\{\theta^e(\mathbf{e}^1_{\mathcal{I}}, \mathbf{e}^2_{\mathcal{J}}) \mid \forall\, E^1_{\mathcal{I}} \in \mathcal{E}^1 \wedge E^2_{\mathcal{J}} \in \mathcal{E}^2\right\}, \quad (22)$$

$$\mathbf{g}^a = \theta^g(\mathbf{g}^1, \mathbf{g}^2), \quad (23)$$

where θ^v, θ^e and θ^g are task-specific functions for merging attributes from both hypergraphs. For the hypergraph matching problems, one may use the predefined node affinity function and edge affinity function as θ^v and θ^e respectively. [5] In this paper, we simply merge attributes by concatenation following [39].

If for each hyperedge ($E^a_{\mathcal{I}\mathcal{J}}$) in a directed association hypergraph, all of its permutation of nodes are also hyperedges in the association hypergraph and all these hyperedges have the same attribute, then the directed association hypergraph can be reduced to an undirected association hypergraph.

Since we have an association hypergraph constructed from the two hypergraphs to be matched, we can turn the hypergraph matching problem into a node selection problem on the association hypergraph. In particular, if a node $V^a_{ij} = \left(V^1_i, V^2_j\right)$ in the association hypergraph is selected, then we recognize that node V^1_i is matched to node V^2_j. This node selection/classification problem can be learned and solved using the proposed HNN-HM.

Moreover, to encode the constraint of the assignment matrix, we add two new types of hyperedges: the row hyperedges and the column hyperedges. According to Eq. (3), each $\mathbf{X}_{ij}$ corresponds to a node V^a_{ij} in the association hypergraph. So for each set of nodes related to the same

[5] From this perspective, a sparse affinity tensor can also be considered as a representation of an association hypergraph.

row or column of the assignment matrix, we add an undirected hyperedge (row hyperedge or column hyperedge) to indicate it (*e.g.* $\left\{V^a_{i_1j_1}, V^a_{i_1j_2}, \ldots, V^a_{i_1j_K}\right\}$ is a row hyperedge). Therefore, for two K-uniform hypergraphs to be matched, the association hypergraph used in this paper is $\mathcal{G}^a = (\mathcal{V}^a, \mathcal{E}^a, \mathcal{R}^a, \mathcal{C}^a, \mathbb{V}^a, \mathbb{E}^a, \mathbb{R}^a, \mathbb{C}^a, \mathbf{g}^a)$, where $\mathcal{R}^a$ and $\mathcal{C}^a$ are row and column hyperedges, and $\mathbb{R}^a$ and $\mathbb{C}^a$ are their attributes. As mentioned in Section 4.1, the proposed HNN-HM framework can handle different types of hyperedges, and each type may have its own update function.

5.2. The encoder module

The encoder module is used to transform attributes in the input space to a new embedding space suitable for the core module. In this module, each attribute is self-updated by a multilayer perceptron (MLP), *i.e.*,

$$\mathbf{r}_l^{a'} = \phi^r_{enc}(\mathbf{r}_l^a) \quad \text{for each association hyperedge,} \tag{24a}$$
$$\mathbf{c}_k^{a'} = \phi^c_{enc}(\mathbf{c}_k^a) \quad \text{for each row hyperedge,} \tag{24b}$$
$$\mathbf{e}_j^{a'} = \phi^e_{enc}(\mathbf{e}_j^a) \quad \text{for each column hyperedge,} \tag{24c}$$
$$\mathbf{v}_i^{a'} = \phi^v_{enc}(\mathbf{v}_i^a) \quad \text{for each association node,} \tag{24d}$$
$$\mathbf{g}^{a'} = \phi^g_{enc}(\mathbf{g}^a) \quad \text{for the global attribute,} \tag{24e}$$

where ϕ^r_{enc}, ϕ^c_{enc}, ϕ^e_{enc}, ϕ^v_{enc}, and ϕ^g_{enc} are different MLPs.

5.3. The core module

The same core module is applied multiple times in the proposed HNN-HM framework. For each step, we merge the output of the encoder and the last output of the core module by concatenating corresponding attribute vectors, and process it using the core module again. The number of steps controls how far the messages are passing among nodes and hyperedges in the hypergraph. In our experiments, we found that ten steps are enough for our task.

We propose two different update strategies: one for the K-uniform directed association hypergraphs and the other for the undirected association hypergraphs.

For the K-uniform directed association hypergraph [6], the update functions (ignore superscript a) are

$$\mathbf{r}'_l = \phi^r_{core}\left(\mathbf{r}_l, \psi^{vr}(\mathbb{V}^r_l), \mathbf{g}\right), \tag{25a}$$
$$\mathbf{c}'_k = \phi^c_{core}\left(\mathbf{c}_k, \psi^{vc}(\mathbb{V}^c_k), \mathbf{g}\right) \tag{25b}$$
$$\mathbf{e}'_j = \phi^e_{core}(\mathbf{e}_j, \mathbf{v}_{E^{(1)}_j}, \mathbf{v}_{E^{(2)}_j}, \ldots, \mathbf{v}_{E^{(K)}_j}, \mathbf{g}), \tag{25c}$$
$$\mathbf{v}'_i = \phi^v_{core}\left(\mathbf{v}_i, \left(\psi^{ev}(\mathbb{E}^{(k)'}_i)\right)_{\forall k}, \psi^{rv}(\mathbb{R}'_i), \psi^{cv}(\mathbb{C}'_i), \mathbf{g}\right), \tag{25d}$$
$$\mathbf{g}' = \phi^g_{core}\left(\mathbf{g}, \psi^{vg}(\mathbb{V}'), \psi^{eg}(\mathbb{E}'), \psi^{rg}(\mathbb{R}'), \psi^{cg}(\mathbb{C}')\right). \tag{25e}$$

The notations used here are similar to those in Eq. (6) with additional labels to distinguish different hyperedges: r for row hyperedges, c for column hyperedges, and e for association hyperedges. In this paper, all update functions (ϕ^r_{core}, ϕ^c_{core}, ϕ^e_{core}, ϕ^v_{core} and ϕ^g_{core}) are different MLPs, and all reduction functions (ψ^{vr}, ψ^{vc}, ψ^{ev}, ψ^{rv}, ψ^{cv}, ψ^{vg}, ψ^{eg}, ψ^{rg} and ψ^{cg}) are the sum reduction function Eq. (14).

For the undirected association hypergraph, the following update functions are used:

$$\mathbf{r}'_l = \phi^r_{core}\left(\mathbf{r}_l, \psi^{vr}(\mathbb{V}^r_l), \mathbf{g}\right), \tag{26a}$$
$$\mathbf{c}'_k = \phi^c_{core}\left(\mathbf{c}_k, \psi^{vc}(\mathbb{V}^c_k), \mathbf{g}\right) \tag{26b}$$
$$\mathbf{e}'_j = \phi^e_{core}\left(\mathbf{e}_j, \psi^{ve}(\mathbf{e}_j, \mathbb{V}_i), \mathbf{g}\right), \tag{26c}$$
$$\mathbf{v}'_i = \phi^v_{core}\left(\mathbf{v}_i, \psi^{ev}(\mathbf{v}_i, \mathbb{E}'_i), \psi^{rv}(\mathbb{R}'_i), \psi^{cv}(\mathbb{C}'_i), \mathbf{g}\right), \tag{26d}$$
$$\mathbf{g}' = \phi^g_{core}\left(\mathbf{g}, \psi^{vg}(\mathbb{V}'), \psi^{eg}(\mathbb{E}'), \psi^{rg}(\mathbb{R}'), \psi^{cg}(\mathbb{C}')\right). \tag{26e}$$

Different from Eq. (25), for the association nodes and the association hyperedges, we use the reduction function Eq. (16) extended with multi-head [34] for ψ^{ve} and ψ^{ev}. By using this reduction function, we can differentiate the contributions from different elements in the same multiset. All update functions and other reduction functions are the same as Eq. (25).

5.4. The decoder module

Contrary to the encoder module, the decoder module is used to extract the desired information from the embedding space. Since we are solving a node classification problem, we only use the node attributes from the hypergraph returned by the core module and update them as follows:

$$\mathbf{v}_i^{a'} = \phi^v_{dec}(\mathbf{v}_i^a), \tag{27}$$

where ϕ_{dec} is an MLP in our experiments.

After that, to approximate the one-to-(at most)-one constraint in a relaxed form, we use a row-wise softmax layer as the output layer, *i.e.*, each row of the prediction assignment matrix is normalized by a softmax function. In testing, the Hungarian algorithm is further used for post-processing.

5.5. Optimization

Since all components of our HNN-HM are differentiable, it can be trained in an end-to-end manner, and all components can be jointly optimized. The assignment matrix indicates whether two nodes from different hypergraphs are matched or not, thus serves as the ground truth for all nodes in the association hypergraph. The loss used in our experiments is binary cross-entropy, and it is optimized by the AMSGrad [29] algorithm with learning rate decay.

6. Experiments

We compare our HNN-HM with seven state-of-the-art third-order algorithms on a synthetic dataset and three real datasets. In particular, we use these algorithms: TM [10] [7],

[6] Remember that it is an association hypergraph, of which each hyperedge has K nodes and the order of these K nodes matters.

[7] We use a bug-fixed version of TM (`https://duchenne.net/publications/code/codeCVPR09fixed.zip`).

Table 1. Accuracy (%) on the Willow object dataset.

(a) Without outliers

Algorithm	Car	Duck	Face	Motor.	Wine.	AVG
TM [10]	69.0	83.3	96.0	82.9	**100**	86.2
IPFP-HM [24]	66.9	77.6	94.7	82.5	**100**	84.3
RRWHM [20]	68.6	79.7	97.6	82.8	**100**	85.7
BCAGM [27]	67.4	77.9	98.1	88.3	**100**	86.3
BCAGM3 [28]	67.3	77.7	97.4	88.6	**100**	86.2
ADGM1 [19]	72.0	76.5	91.9	90.7	98.9	86.0
ADGM2 [19]	72.2	77.4	94.5	91.9	99.3	87.1
HNN-HM	**91.5**	**92.3**	**100**	**99.9**	**100**	**96.8**

(b) With outliers (#outliers=5)

Algorithm	Car	Duck	Face	Motor.	Wine.	AVG
TM [10]	40.3	48.2	56.2	50.1	91.7	57.3
IPFP-HM [24]	40.6	47.3	60.1	49.3	91.9	57.9
RRWHM [20]	41.6	50.0	64.5	53.3	91.5	60.2
BCAGM [27]	42.3	49.0	69.0	54.5	**94.6**	61.9
BCAGM3 [28]	41.8	49.1	68.4	54.4	94.5	61.6
ADGM1 [19]	41.9	46.4	46.9	50.6	86.1	54.4
ADGM2 [19]	41.6	45.7	52.8	52.0	85.8	55.6
HNN-HM	**76.7**	**72.9**	**97.5**	**85.6**	93.3	**85.2**

Table 2. Accuracy (%) on the Pascal VOC dataset.

Algorithm	aero	bike	bird	boat	botl	bus	car	cat	chai	cow	dtab	dog	hors	mbk	prsn	plnt	shp	sofa	trn	tv	AVG
TM [10]	29.1	35.5	42.6	43.3	**93.2**	32.7	40.8	39.8	31.9	36.5	44.7	34.2	35.7	34.3	31.7	65.0	39.3	45.6	75.1	39.8	43.5
IPFP-HM [24]	26.9	34.6	41.4	42.1	87.0	32.8	40.0	37.8	30.9	34.9	50.2	32.4	33.7	33.6	30.2	63.4	37.0	44.5	72.8	38.1	42.2
RRWHM [20]	28.0	36.2	42.7	43.3	91.1	33.8	40.9	38.6	32.2	35.6	49.2	34.0	35.9	34.5	31.4	63.2	38.9	46.5	75.3	39.9	43.6
BCAGM [27]	28.0	38.1	41.7	44.3	90.2	36.1	41.5	39.4	32.8	35.4	49.3	33.6	34.4	34.8	31.8	63.0	39.5	47.3	76.8	39.9	43.9
BCAGM3 [28]	28.3	38.6	41.9	45.0	90.3	36.1	42.3	38.6	33.5	35.7	51.4	33.6	34.0	35.0	31.1	62.6	39.1	47.9	77.5	39.6	44.1
ADGM1 [19]	26.5	38.5	37.3	38.2	91.4	38.3	38.6	35.4	31.8	35.9	53.3	32.9	33.3	35.2	30.3	63.4	34.9	51.3	45.2	29.5	41.1
ADGM2 [19]	26.8	38.5	38.1	39.6	91.1	40.0	38.4	35.6	31.4	36.1	54.1	33.3	32.5	34.8	29.5	63.6	35.0	50.5	57.0	31.0	41.8
HNN-HM	**39.6**	**55.7**	**60.7**	**76.4**	87.3	**86.2**	**77.6**	**54.2**	**50.0**	**60.7**	**78.8**	**51.2**	**55.8**	**60.2**	**52.5**	**96.5**	**58.7**	**68.4**	**96.2**	**92.8**	**68.0**

IPFP-HM (the algorithm 3 in [24]), RRWHM [20], the best two in the BCAGM variants: BCAGM+MP [27] (BCAGM for short) and Adapt-BCAGM3+MP [28] (BCAGM3 for short), and two variants of the ADGM [19] algorithm: ADGM1 and ADGM2. The recommended or default hyperparameters are used in our experiments. To obtain a proper assignment matrix, we use the Hungarian algorithm to discretize the output of all methods.

Following [10, 20, 27], we only use geometric information (*i.e*. the coordinates of each point) to construct hypergraphs for matching and evaluate all algorithms on the third-order hypergraph matching problems. The node affinity is set to zero. And the hyperedge affinity (used by other algorithms, not by ours) is computed by

$$d_{\mathcal{I}\mathcal{J}} = \exp(\frac{1}{\gamma}\left\|\mathbf{e}^1_{\mathcal{I}} - \mathbf{e}^2_{\mathcal{J}}\right\|_2) \tag{28}$$

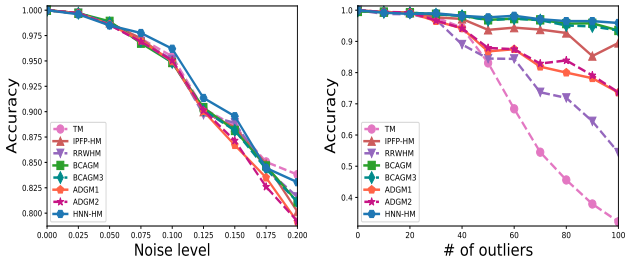
where γ is the mean of all distances, and $\mathbf{e}^1_{\mathcal{I}}$ and $\mathbf{e}^2_{\mathcal{J}}$ are the hyperedge attributes [8] computed following [10].

The association hypergraph used in our algorithm is constructed in the same way as RRWHM [20]. But instead of using the affinity (derived from the point coordinates) as attributes, we directly use the point coordinates as attributes for our algorithm. The initial values of the global attribute vector, the row attribute vectors, and the column attribute vectors are set to zero in our experiments.

Just like [10, 19, 20, 27, 28], we follow the same sampling strategy proposed in TM [10] to reduce the number of non-zero values in the affinity tensor, which in fact reduces the number of association hyperedges. To further reduce the number of association hyperedges, we also convert directed association hyperedges to undirected hyperedges and average their outputs from the encoder to get order-insensitive attributes. For example, all the following six association hyperedges: $(V^a_{11}, V^a_{22}, V^a_{33})$, $(V^a_{11}, V^a_{33}, V^a_{22})$, $(V^a_{22}, V^a_{11}, V^a_{33})$, $(V^a_{22}, V^a_{33}, V^a_{11})$, $(V^a_{33}, V^a_{11}, V^a_{22})$, and $(V^a_{33}, V^a_{22}, V^a_{11})$, will be converted to one undirected hyperedge $\{V^a_{11}, V^a_{22}, V^a_{33}\}$, and the average of all their outputs from the encoder will be regarded as the attribute vector of the new undirected hyperedge. This makes sense if we consider each of their associated attribute vectors as an affinity feature vector between two triplets of nodes: (V^1_1, V^1_2, V^1_3) and (V^2_1, V^2_2, V^2_3), thus we can expect that all these attribute vectors should be the same. After this operation, all hyperedges of the association hypergraph are undirected. Hence we can use the update functions for the undirected association hypergraph in the core module. On the Willow dataset, we have tried both the undirected and directed versions and found that their accuracies are comparable: 96.8% (undirected) and 96.5% (directed). Since the undirected version is memory-efficient, we prefer it in all our experiments.

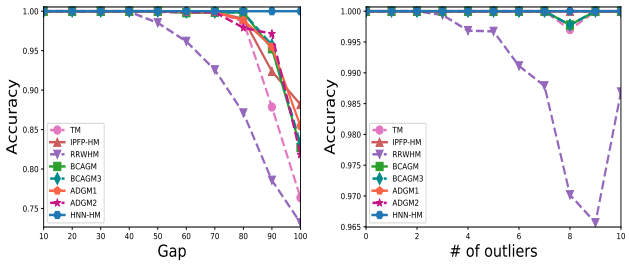
Synthetic dataset We first evaluate our algorithm on matching 2D point sets. Following [27], we randomly sample n_{inlier} points from the Gaussian distribution $\mathcal{N}(0, 1)$ as the nodes in the first hypergraph and obtain nodes in another hypergraph by scaling these points, adding Gaussian noise taken from $\mathcal{N}(\sigma, 1)$, and appending $n_{outlier}$ points drawn from $\mathcal{N}(0, 1)$ as outliers. Hyperedges of the first hypergraph are randomly generated by choosing triplets of

[8] Each hyperedge attribute is a vector concatenated by the sine values of the three angles of the corresponding triangle (constructed by the triplet of points in that hyperedge).



(a) varying noise level (b) varying outliers

Figure 4. Evaluation on the synthetic dataset.



(a) varying gap (#outliers=10) (b) varying outliers (gap=50)

Figure 5. Evaluation on the CMU house dataset.

nodes, and hyperedges in the second hypergraph are fully connected. After that, the sampling strategy in TM [10] is applied to reduce non-zero values in the affinity tensor.

Two settings of experiments are conducted. One is for testing the robustness against noise (controlled by σ), so we set $n_{inlier} = 20$ and vary the noise level σ from 0 to 0.2, without scaling or outliers. Another one is for the evaluation of stability to outliers, thus we set $n_{inlier} = 10$, and increase the number of outliers $n_{outlier}$ from 0 to 100, with 1.5 times the scaling and $\sigma = 0.03$. The results in Fig. 4 (each curve is averaged over 100 runs) show that our method performs well on noise and outliers and is similar to or better than the state-of-the-art methods.

CMU house dataset The CMU house dataset is popular for evaluating hypergraph matching algorithms [10, 19, 20, 27, 28]. It consists of a sequence 111 frames taken from the same moving toy house. The larger the sequence gap between two frames, the more deformation of the house object. Each frame is labeled with 30 keypoints. To simulate the outlier scenario, we keep all 30 points in one frame and remove $n_{outlier}$ points in another frame. For training our method, we sample 23 frames uniformly from the dataset.

The algorithms are evaluated in two settings: the deformation setting and the outlier setting. For the deformation setting, we choose 20 points in the first frame and all 30 points in the second frame (*i.e.* $n_{outlier} = 10$), and increase the sequence gap from 10 to 100 with the step size of 10. Results are averaged over all frame pairs that have the same sequence gap. For the outlier setting, we set the sequence gap to 50, and increase the value of $n_{outlier}$ from 0 to 10. Again, results are averaged over all frame pairs that have the same sequence gap (gap=50). In this dataset, the accuracy of our HNN-HM is nearly 100% (see Fig. 5), which shows its power over other state-of-the-art algorithms.

Willow object dataset The Willow object dataset [6] contains five categories, each with at least 40 images, and each image is labeled with ten landmarks. We select 20 images per category for training and let the remaining for testing. All algorithms are evaluated on 1000 pairs of images (randomly selected from the test set) per category. To test their performance under outliers, we use SIFT to detect five keypoints and append them as outliers to the ten labeled points.

Results on the Willow object dataset are shown in Table 1. Without any outliers (see Table 1a), ADGM2 [19] is superior to its predecessors. With five outliers (see Table 1b), BCAGM [27] performs quite well for its usage of MPM [8] which is robust to outliers. Nevertheless, our method significantly outperforms all rivals in both cases.

Pascal VOC dataset The Pascal VOC dataset [11] with labeled keypoints [3] has 20 classes and is challenging for its variety of scale, pose, and illumination. Following [36], we use the filtered dataset (7020 training and 1682 test images). For evaluation, we randomly select 1000 image pairs (at least three common points) from each class in the test set. Table 2 summarize the results. Except for the bottle class, our method surpasses almost all other methods, which clearly shows the advantage of our method.

7. Conclusion

In this paper, we propose the HNN-HM algorithm, which is the first unified hypergraph neural network (HNN) solution for hypergraph matching. We first convert the hypergraph matching problem into a node classification problem on an association hypergraph and then develop an HNN solution to solve the node classification problem. Experiment results clearly show the advantage of our method over state-of-the-art hypergraph matching algorithms.

Acknowledgement. We sincerely thank anonymous reviewers for their insightful comments. Yong Xu thanks the supports of the National Nature Science Foundation of China (62072188) and the Science and Technology Program of Guangdong Province (2019A050510010).

References

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, 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.

[2] Serge J. Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24(4):509–522, 2002.

[3] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[4] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, 2011.

[5] Tibério S. Caetano, Julian J. McAuley, Li Cheng, Quoc V. Le, and Alex J. Smola. Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(6):1048–1058, 2009.

[6] Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[7] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *European Conference on Computer Vision*, 2010.

[8] Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[9] Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In *Advances in Neural Information Processing Systems*, 2006.

[10] Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(12):2383–2395, 2011.

[11] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[12] Matthias Fey, Jan Eric Lenssen, Christopher Morris, Jonathan Masci, and Nils M. Kriege. Deep graph matching consensus. In *International Conference on Learning Representations (ICLR)*, 2020.

[13] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] Pasquale Foggia, Gennaro Percannella, and Mario Vento. Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(1), 2014.

[15] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993.

[16] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 18(4):377–388, 1996.

[17] Arthur Ardeshir Goshtasby. *2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications*. John Wiley & Sons, 2005.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[19] D. Khuê Lê-Huu and Nikos Paragios. Alternating direction graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] Jungmin Lee, Minsu Cho, and Kyoung Mu Lee. Hyper-graph matching via reweighted random walks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[21] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *IEEE International Conference on Computer Vision (ICCV)*, 2005.

[22] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and MAP inference. In *Advances in Neural Information Processing Systems*, 2009.

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

[24] Marius Leordeanu, Andrei Zanfir, and Cristian Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[25] Zhi-Yong Liu and Hong Qiao. GNCCP—graduated nonconvexityand concavity procedure. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(6):1258–1267, 2014.

[26] David G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, 1999.

[27] Quynh Nguyen Ngoc, Antoine Gautier, and Matthias Hein. A flexible tensor block coordinate ascent scheme for hypergraph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[28] Quynh Nguyen, Francesco Tudisco, Antoine Gautier, and Matthias Hein. An efficient multilinear optimization framework for hypergraph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(6):1054–1075, 2017.

[29] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations (ICLR)*, 2018.

[30] 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*, 2020.

[31] Xinchu Shi, Haibin Ling, Weiming Hu, Junliang Xing, and Yanning Zhang. Tensor power iteration for multi-graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] Xinchu Shi, Haibin Ling, Junliang Xing, and Weiming Hu. Multi-target tracking by rank-1 tensor approximation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[33] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *European Conference on Computer Vision*, 2008.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[35] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations (ICLR)*, 2020.

[36] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[37] 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. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[38] Tao Wang, Haibin Ling, Congyan Lang, and Songhe Feng. Graph matching with adaptive and branching path following. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(12):2853–2867, 2018.

[39] Tao Wang, He Liu, Yidong Li, Yi Jin, Xiaohui Hou, and Haibin Ling. Learning combinatorial solver for graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[40] Junchi Yan, Changsheng Li, Yin Li, and Guitao Cao. Adaptive discrete hypergraph matching. *IEEE Transactions on Cybernetics*, 48(2):765–779, 2018.

[41] Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and Hungarian attention. In *International Conference on Learning Representations (ICLR)*, 2020.

[42] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[43] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(12):2227–2242, 2009.

[44] Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[45] Feng Zhou and Fernando De la Torre. Factorized graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.