Contrastive General Graph Matching with Adaptive Augmentation Sampling Appendix

Appendix A Implementation Details

Graph Encoder

The initial node features of the input graph $\mathcal{G}$ is denoted as $\mathbf{X} \in \mathbb{R}^{N\times F}$, are first projected into a lower dimension $\mathbf{H^0} = \mathbf{W}_{\text{init}}\mathbf{X} \in \mathbb{R}^{N\times F'}$ where $F' \leq F$. We employ GraphSAGE [Hamilton *et al.*, 2018] with the mean aggregator as shown below:

$$\hat{\mathbf{h}}_v^k = \sigma\Big(\mathbf{W}_{\text{GS}}^k \,\text{MEAN}\left(\{\hat{\mathbf{h}}_v^{k-1}\} \cup \{\hat{\mathbf{h}}_u^{k-1}, \forall u \in \mathcal{N}(v)\}\right) + \sum_{i=1}^{k-1} \hat{\mathbf{h}}_v^{k-1}\Big) \tag{1}$$

where $\hat{\mathbf{h}}_v^k$ denotes the k-th layer embedding of node v, $\mathbf{W}_{\text{GS}}^k$ is the k-th layer learned weight matrix, σ denotes a non-linear activation function, and $\mathcal{N}(v)$ is the set of neighbors of node v.

For node representation, we concatenate the embeddings from all the GCN layers and project them using a two-layer Multi-Layer Perceptron (MLP). This yields the final node representation $\widehat{\mathbf{H}} \in \mathbb{R}^{N\times F''}$.

$$\hat{\mathbf{h}}_v = \mathbf{W}_{\text{proj2}}\sigma\left(\mathbf{W}_{\text{proj1}} \oplus_{k=1}^{L} \hat{\mathbf{h}}_v^k + \mathbf{b}_{\text{proj1}}\right) + \mathbf{b}_{\text{proj2}} \tag{2}$$

where $\hat{\mathbf{h}}_v$ denotes the final representation of node v, $\oplus_{k=1}^{L}\hat{\mathbf{h}}_v^k$ represents the concatenation of the embeddings from all the L layers of the encoder for node v, $\mathbf{W}_{\text{proj1}}$ and $\mathbf{W}_{\text{proj2}}$ are learned weight matrices, $\mathbf{b}_{\text{proj1}}$ and $\mathbf{b}_{\text{proj2}}$ are bias terms.

Finally, the global representation $\widehat{\mathbf{h}}_{\mathcal{G}}$ is attained by a readout function:

$$\widehat{\mathbf{h}}_{\mathcal{G}} = \text{AGGR}\Big(\{\mathbf{W}_{\text{read2}}\sigma\left(\mathbf{W}_{\text{read1}} \oplus_{k=1}^{L} \hat{\mathbf{h}}_v^k + \mathbf{b}_{\text{read1}}\right) + \mathbf{b}_{\text{read2}}, \forall v \in V\}\Big) \tag{3}$$

here AGGR is an aggregator function that consolidates node embeddings using methods like mean, sum, or max pooling. $\mathbf{W}_{\text{read1}}$, $\mathbf{W}_{\text{read2}}$, $\mathbf{W}_{\text{read1}}$, and $\mathbf{W}_{\text{read2}}$ are learnable weights. And V is the entire set of nodes in $\mathcal{G}$.

Algorithm

The psudocode of training the proposed GCGM with our BiAS strategy is listed in Alg. 1. We first instantiate a pool of random augmentation pairs $\mathcal{P}$, each having an initial weight w_i^0 set to e^{α}, along with hyperparameters α and λ for BiAS. During the training, for each mini-batch t, we compute a probability distribution P_t for all augmentation pairs based on their current weights. Then, for every graph $\mathcal{G}$ in our batch of graphs, we sample an augmentation pair using P_t and apply these augmentations to obtain two new augmented graphs $\tilde{\mathcal{G}}^A$ and $\tilde{\mathcal{G}}^B$. We then utilize the GM solver adopted to predict the matching matrix between these augmented views and attain the performance score i.e., F_1 score. After processing the entire batch, we update the weights of our augmentation pairs by first computing ϕ_t^i, the mean performance score for all matchings where augmentation pair (τ_j, τ_k) was previously applied up to the current mini-batch t, and then updating weights. Subsequently, we update our pool $\mathcal{P}$ with these new weights. We continue this process for each mini-batch until the termination criteria are met, resulting in a trained model.

Algorithm 1 Training of GCGM with BiAS

Require: Augmentation pairs pool $\mathcal{P}$, Initial weight for each augmentation pair $w_i^0 = e^{\alpha}$, Hyperparameters α, λ

```
for t in training steps do
    P_t(i) = w_t^i / Σ_{j∈S} w_t^j, ∀i ∈ |P|   # probability distribution
    for each graph G in batch of graphs {G_1, ..., G_N} do
        (τ_j, τ_k) ~ P_t over P   # sample augmentation pairs
        G̃^A ← τ_j(G), G̃^B ← τ_k(G)   # augment graph
        F_1 ← Match between G̃^A and G̃^B
    end for
    w_i^{t+1} ← λ · w_i^t + (1 − λ) · e^{α·(1−φ_t^i)}   # update weights
    Update P with the new weights
end for
```

Ensure: Trained model

Model Configurations

We pre-train our model using the Adam Optimizer. Detailed hyperparameters used could be found in Tab. A.

Hardware Configuration

For our experiments, we utilized a Linux machine equipped with an AMD EPYC 7742 64-Core CPU (2.25GHz) paired with a GeForce RTX 3090 GPU.

GCGM	Pascal VOC		Willow		SPair-71k		Synthetic
	BBGM	NGMv2	BBGM	NGMv2	BBGM	NGMv2	NGMv2
batch size	32	32	16	16	32	32	32
learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-5}	1×10^{-4}	1×10^{-5}
weight decay	1×10^{-3}	1×10^{-5}	1×10^{-4}	1×10^{-4}	1×10^{-5}	1×10^{-5}	1×10^{-2}
epoch	10	10	10	10	10	10	10
hidden dim	256	1024	256	1024	1024	1024	16
output dim	256	1024	256	1024	1024	1024	8
# layer	2	2	2	2	2	2	1
activation	leaky relu	relu	elu	elu	prelu	relu	leaky relu
aggregation	mean	mean	max	mean	mean	mean	mean
readout	mean	mean	max	max	add	mean	max
temperature	0.07	0.07	0.07	0.07	0.07	0.07	0.07

Table A: Hyperparameter configurations.

Type	Feature	Structure	Matching Scenario	Hyperparameters
ND	✗	✓	outliers	$p_{nd} \in [0.1, 0.9]$: fraction of nodes dropped
EA	✗	✓	new connections	$p_{ea} \in [0.1, 0.9]$: fraction of edges added
FM	✓	✗	variations	$p_{fm} \in [0.1, 0.9]$: probability of masking each feature dimension
Mixup	✓	✗	variations	$\gamma \in [0.1, 0.9]$: mixup rate

Table B: Details of the remaining four graph augmentation types.

Appendix B Graph Augmentations

In addition to the four major types of augmentations, we have also included the following four. Details can be found in Tab. B.

Node Dropping (ND). In the context of graph matching, recognizing outliers is crucial. While inserting dummy nodes introduces direct outliers, an equally effective strategy is to create outliers indirectly by dropping existing nodes. When random nodes are dropped from one view, their corresponding nodes in the other view automatically become outliers.

Edge Addition (EA). Though real-world graphs are often sparse, additional connections can emerge. These new edges can significantly alter the graph's structure and information flow. With the edge addition strategy [Zeng and Xie, 2021], random edges are added between nodes that are not directly connected but with a path between them, thus altering the original connectivity.

Feature Masking (FM). Node features in graphs can vary widely or be incomplete. Feature masking, a standard graph augmentation, addresses this by randomly masking certain features as zero, ensuring models learn robust representations. Similar to feature scaling, it helps counteract feature variations and prevents over-reliance on specific features.

Mixup. We adopt the SCGM [Liu *et al.*, 2022] approach to include label mixup as an additional feature-space augmentation. However, there's a distinct difference. Whereas SCGM calculates the contrastive loss immediately after applying image augmentations (without the mixup influencing the contrastive loss computation), we treat mixup as standard graph augmentation and then calculate our node-level contrastive loss.

Appendix C Dataset Preparation

Real-world Dataset

We strictly followed the configurations specified by each method during experiments. For preprocessing and extracting the initial node features, we followed the approach of SCGM [Liu *et al.*, 2022]. Images underwent standard processing and were resized to 256×256. Using VGG16 [Simonyan and Zisserman, 2014], which was pre-trained on ImageNet [Deng *et al.*, 2009], we applied bilinear interpolation to each keypoint to obtain the node features. The graph was constructed using Delaunay triangulation. And the features of the connected nodes, encoded by our graph encoder, were concatenated to form their respective edge features. For evaluations on Pascal VOC [Bourdev and Malik, 2009; Everingham *et al.*, 2010] and SPair-71k [Min *et al.*, 2019], we sampled 1000 random graph pairs from each class. For Willow Object dataset [Cho *et al.*, 2013], we sampled 100 graph pairs per category.

Synthetic Dataset

Following the previous work [Wang *et al.*, 2021; Liu *et al.*, 2023], we generated ten sets of random points on the 2D plane, with coordinates sampled from $U(0,1) \times U(0,1)$. These sets served as our ground truth. The points underwent distortion via random scaling from $U(1-\delta_s, 1+\delta_s)$ with $\delta_s = 0.2$, and had noise $N\left(0, \sigma_n^2\right)$ added to their position, where $\sigma_n = 0.02$. Additionally, up to two outliers were added to each graph. From each ground truth set, we derived 200 graphs for training and 100 for testing, leading to total 2000 training and 1000 testing samples. And we also follow a 80:20 train-validation split. During evaluation, 1000 random graph pairs were sampled from each ground-truth set.

Appendix D Baselines

We selected eight baselines for our study, which includes supervised methods CIE [Yu *et al.*, 2019], BBGM [Rolínek *et al.*, 2020], and NGMv2 [Wang *et al.*, 2021]. In addition, we chose learning-free methods: RRWM [Cho *et al.*, 2010], IFPF [Leordeanu *et al.*, 2009], and SM [Leordeanu and Hebert, 2005]. For self-supervised baselines, our comparison were made against GANN-GM [Wang *et al.*, 2023] and SCGM [Liu *et al.*, 2022].

We utilized the implementations maintained by [Wang *et al.*, 2021], which have been well-received in the GM domain. To ensure a fair comparison, we adopted their optimal configurations for the Pascal VOC and Willow Object datasets. Additionally, we maintained the SPair-71k's configuration identical to that used for the Pascal VOC, a practice we also followed.

Appendix E Performance Metric

Given a pair of input graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with N_1 and N_2 nodes respectively, we derive the predicted matching matrix $\hat{\mathbf{G}} \in \{0,1\}^{N_1 \times N_2}$ from the GM backbone, Then we calculate the accuracy/recall and precision between the prediction and the ground-truth permutation matrix $\mathbf{G}^{gt} \in \{0,1\}^{N_1 \times N_2}$.

Methods	Synthetic Intsec	Unfilt
CIE [SUP]	12.2 ± 5.2	-
BBGM [SUP]	79.0 ± 2.4	60.2 ± 3.7
NGMv2 [SUP]	82.6 ± 1.8	62.5 ± 2.6
IPFP	68.1 ± 0.02	48.8 ± 0.07
RRWM	80.9 ± 0.04	60.9 ± 0.12
SM	64.3 ± 0.07	43.4 ± 0.06

Table C: Average performance of supervised and learning-free methods on the Synthetic Dataset.

$$\text{Recall} = \frac{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} \hat{g}_{ij} \cdot g_{ij}^{gt}}{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} g_{ij}^{gt}} \tag{4}$$

$$\text{Precision} = \frac{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} \hat{g}_{ij} \cdot g_{ij}^{gt}}{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} \hat{g}_{ij}} \tag{5}$$

$$F_1 = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{6}$$

In the *Intersection* setting, recall is equal to both precision and F_1 score.

Appendix F Additional Model Analyses

Performance of Supervised and Learning-Free Methods on the Synthetic Dataset

In this subsection, we present additional results for supervised and learning-free methods on the Synthetic dataset in Tab. C. Supervised methods require significant groundtruth matching labels, while learning-free methods apply a heuristic based on graph structure/affinity matrix. Similar to Tab. 2 in the paper, supervised methods perform better due to their access to labels. Meanwhile, learning-free methods also perform well on Synthetic data which exhibit highly consistent structural patterns that can be exploited by their heuristics, while avoiding overfitting to node features sampled from a uniform distribution. However, they underperform on real-world data where noise and outliers are prevalent.

Detailed Performance by Class

As shown in Tab. D, GCGM consistently showcase competitive performance compared to other unsupervised methods on Pascal VOC dataset. In several categories like ‘bottle’, ‘table’, and ‘plant’, GCGM not only takes the lead but does so with a significant advantage, comparable to supervised methods. However, when the SCGM achieves higher scores, the difference with GCGM is relatively modest. Meanwhile, IPFP [Leordeanu *et al.*, 2009], RRWM [Cho *et al.*, 2010], SM [Leordeanu and Hebert, 2005], and GANN-GM [Wang *et al.*, 2023] generally perform at a noticeably lower level compared to both GCGM and SCGM. Tab. E compares the performance of GCGM with other methods on Willow Object dataset. GCGM, when coupled with either BBGM [Rolínek *et al.*, 2020] or NGMv2 [Wang *et al.*, 2021], consistently exhibits superior performance across the majority of categories. Notably, in ‘car’, ‘duck’, and ‘winebottle’, the GCGM + NGMv2 achieves the highest scores, surpassing even some supervised approaches such as CIE [Yu *et al.*, 2019] and NGMv2.

Comparison of GCGM vs. SCGM: Varied Information Levels

To illustrate GCGM’s ability to achieve outstanding performance with minimal information (specifically, only the graph) in contrast to SCGM’s two-stage augmentations which requires access to additional image features, we conducted experiments on three real-world datasets. In these experiments, we excluded image augmentation and visual backbone fine-tuning (SCGM’s default training configuration) from SCGM. The results presented in Tab. G show that when restricted to only graph augmentation (although SCGM still utilizes the image feature to aid affinity learning), SCGM’s performance declines significantly across three datasets. Intriguingly, when the visual backbone of SCGM is frozen, its performances on Pascal VOC and SPair-71k datasets actually improve compared to when fine-tuning is permitted. This could be due to feature redundancy from image augmentation, where multiple transformations activate similar features. Additionally, some augmentations might introduce noise or irrelevant variations, making further fine-tuning counterproductive. In summary, our proposed GCGM consistently outperforms others, relying solely on graph information. This underscores its effectiveness and efficiency in managing real-world datasets.

Effect of BiAS Design

The ablation study, presented in Table. F, evaluates the overall effectiveness of the complete BiAS scheme against configurations where individual design elements such as momentum update and ϕ_t^i are excluded, across multiple datasets. We use uniform sampling as our baseline denoted as ‘Uniform’, establishing a fundamental performance reference. The ‘ϕ^i’ signifies that BiAS updates the weight of augmentation pair i based solely on its current mini-batch performance, as opposed to averaging over all previous matchings, which would lead to less smoother weight updates. Although the outcomes are largely similar, subtle variations are observed across datasets. Notably, by turning off the momentum update and combining with ϕ^i (labeled as ‘$\lambda = 0 \wedge \phi^i$’), we see slight improvements in specific situations, particularly in the *Unfiltered* settings of the SPair-71k and Synthetic datasets. This suggests that promoting the most challenging augmentations can occasionally yield superior results, especially in the presence of outliers. Nevertheless, the comprehensive BiAS approach consistently outperforms in most settings across datasets, reinforcing its overall robustness and highlighting the importance of both the performance metric ϕ_t^i and momentum update.

Varying Size of Augmentation Pool

In Fig. A, we present GCGM’s performance on the Pascal VOC dataset using both the BiAS and ‘Uniform’ samplers across varying augmentation pool sizes $|\mathcal{P}|$ ranging from 2 to 1024. Both sampler benefit from employing a larger augmentation pool, leading to improved and more consistent scores.

Methods	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	motorbike	person	plant	sheep	sofa	train	tvmonitor
BBGM (SUP)	38.5	65.2	51.6	37.5	85.6	62.1	25.7	56.9	36.7	58.4	44.0	54.1	56.1	61.7	32.7	95.8	49.1	31.8	73.3	82.8
NGMv2 (SUP)	44.0	65.6	52.6	43.4	86.2	60.0	43.5	59.0	39.3	57.2	36.7	55.2	56.3	61.9	41.5	94.3	48.1	36.9	70.7	81.9
IPFP	20.9	42.1	27.1	23.6	42.5	32.9	21.4	31.4	20.2	25	37	25.8	27.6	32.8	16.8	56	22	15.2	45.3	64.8
RRWM	22.0	43.4	28.0	23.5	45.5	31.7	20.3	31.8	21.0	25.9	31.5	26.0	30	32.6	18.1	57.5	22.5	15.2	44.2	63.1
SM	19.7	41.2	25.4	22.5	43.2	32.0	20.5	30.6	18.9	23.5	32.9	25.3	25.3	31.9	16.7	53.3	20.3	14.5	43.9	66.1
GANN-GMˆ	13.2	22.1	15.9	19.6	37.1	31.3	17.2	15.6	19.8	16.2	23.7	13.1	15.2	19.8	13.2	37.6	16.5	17.3	37.3	67.0
SCGM + BBGM	**23.5**	**48.5**	**34.2**	29.9	59.6	**33.9**	**22.5**	**33.8**	22.5	**30.7**	25.3	**28.3**	**36**	**38.4**	19.8	76.2	**27.8**	21.8	**52**	66.5
SCGM + NGMv2	21.6	43.3	30.9	25.9	54.2	33.1	21.8	29.4	23.1	26.9	22.5	24.6	30.2	35.5	20.1	57.1	24.8	20.9	46.6	65.5
GCGM + BBGM	17.5	38.6	26.3	**30.4**	**76.4**	32.8	21.1	26.2	**26.2**	23	**45.4**	21	26.3	30.5	20.9	**93**	19.8	24.2	47.8	76.2
GCGM + NGMv2	20.0	39.9	28.7	30	74.7	32.4	21.7	32.7	25.5	26.1	42.9	25	29.5	33.2	**21.2**	91.7	22.3	**25.9**	48	**76.4**

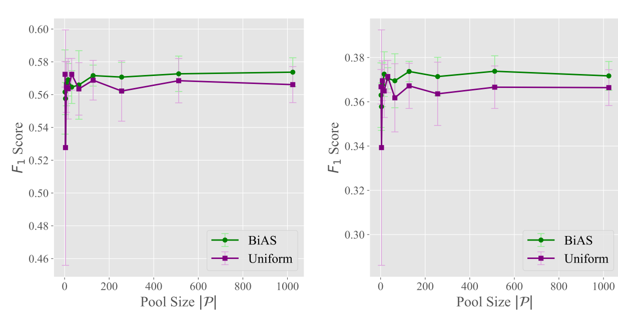
Table D: Average performance w.r.t F_1 score (%) by class on the Pascal VOC dataset's *Unfiltered* setting. Supervised methods are annotated with 'SUP'. ˆ: unsupervised methods that require categorical information. Bold/underlined: best/runner-up results.

Methods	car	duck	face	motorbike	winebottle
CIE (SUP)	73.7	72.2	98.1	87.5	81.7
BBGM (SUP)	95.6	87.7	100	99.3	98.2
NGMv2 (SUP)	93.5	85.3	99.9	97.5	96.1
IPFP	76.3	69.6	99.8	69.7	85.1
RRWM	78.6	73.3	100	77.2	87.7
SM	76.7	72.5	99.9	71.4	86.1
GANN-GMˆ	84.5	85.2	100	81.8	95.3
SCGM + BBGM	88.4	86.3	100	92.8	98.1
SCGM + NGMv2	79.4	73.0	99.2	81.2	88.1
GCGM + BBGM	93.0	85.9	100	**94.6**	98.5
GCGM + NGMv2	**95.7**	**86.6**	100	93.6	**99.1**

Table E: Average performance by class on Willow Object dataset.

Configurations	Pascal VOC		Willow	SPair-71k		Synthetic	
	Intsec	Unfilt	Intsec	Intsec	Unfilt	Intsec	Unfilt
Uniform	56.9	36.7	94.7	62.0	34.8	57.5	40.0
ϕ^i	56.7	36.7	94.9	61.0	33.7	57.9	40.2
$\lambda = 0 \wedge \phi^i$	56.5	37.0	94.9	61.2	**35.5**	58.0	**40.3**
BiAS	**57.3**	**37.4**	**95.0**	**62.6**	35.4	**58.1**	39.9

Table F: Ablation study on BiAS design.



(a) Intersection

(b) Unfiltered

Figure A: Comparison of BiAS and 'Uniform' samplers across different augmentation pool sizes on Pascal VOC dataset.

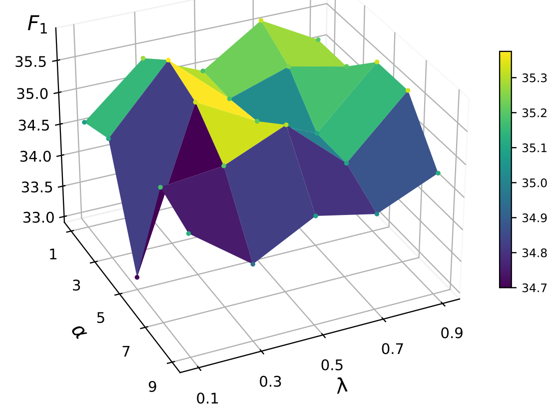


Figure B: Sensitivity of BiAS to different hyperparameter settings on SPair-71k dataset.

Notably, BiAS consistently surpasses the 'Uniform' sampler in both the *Intersection* and *Unfiltered* scenarios.

Hyperparameter Sensitivity

To investigate the sensitivity of BiAS's hyperparameters, specifically λ and α, we conducted experiments using various combinations of these parameters on the SPair-71k dataset. As shown in Fig. B, we trained GCGM + BiAS for $\lambda \in [0.1, 0.9]$ and $\alpha \in [1, 10]$, and tested under *Unfiltered* setting (each combination was run 5 times with different random seeds and $|\mathcal{P}| = 512$). From the results, we observe that larger values of α can lead to sub-optimal performance. Extremely high α values create a skewed distribution of the weights for augmentation pairs, increasing the likelihood of sampling challenging augmentations. This reduces data diversity and can hinder performance. For λ, values at the extremes should be avoided to ensure a smooth weight updating process. Nevertheless, even with extreme parameter settings, BiAS tends to produce good results, indicating its robustness and effectiveness in adaptively sampling from a large pool of randomly instantiated augmentation pairs with just two hyperparameters.

Methods	Fine-tune Visual Backbone	Image Augmentation	Graph Augmentation	Pascal VOC Intsec	Pascal VOC Unfilt	Willow Intsec	SPair-71k Intsec	SPair-71k Unfilt
SCGM + BBGM	✓	✓	✓	54.8	36.6	93.1	60.2	34.1
	✗	✓	✓	55.7	36.1	91.5	62.6	33.4
	✗	✗	✓	53.8	32.4	84.9	46.4	20.0
SCGM + NGMv2	✓	✓	✓	50.8	32.9	84.2	59.8	30.5
	✗	✓	✓	55.5	34.4	77.2	59.5	28.8
	✗	✗	✓	47.9	30.2	77.7	56.8	25.3
GCGM + BBGM	✗	✗	✓	56.8	36.2	94.4	60.6	**35.9**
GCGM + NGMv2	✗	✗	✓	**57.3**	**37.4**	**95.0**	**62.6**	35.4

Table G: Comparison of GCGM and SCGM based on access to different levels of information across three real-world datasets.

References

[Bourdev and Malik, 2009] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *International Conference on Computer Vision*, pages 1365–1372. IEEE, 2009.

[Cho *et al.*, 2010] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V 11*, pages 492–505. Springer, 2010.

[Cho *et al.*, 2013] Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *International Conference on Computer Vision*, pages 25–32, 2013.

[Deng *et al.*, 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[Everingham *et al.*, 2010] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2010.

[Hamilton *et al.*, 2018] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[Leordeanu and Hebert, 2005] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1482–1489. IEEE, 2005.

[Leordeanu *et al.*, 2009] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. *Advances in neural information processing systems*, 22, 2009.

[Liu *et al.*, 2022] Chang Liu, Shaofeng Zhang, Xiaokang Yang, and Junchi Yan. Self-supervised learning of visual graph matching. In *European Conference on Computer Vision*, pages 370–388. Springer, 2022.

[Liu *et al.*, 2023] Chang Liu, Zetian Jiang, Runzhong Wang, Lingxiao Huang, Pinyan Lu, and Junchi Yan. Revocable deep reinforcement learning with affinity regularization for outlier-robust graph matching. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[Min *et al.*, 2019] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Spair-71k: A large-scale benchmark for semantic correspondence. *arXiv prepreint arXiv:1908.10543*, 2019.

[Rolínek *et al.*, 2020] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 407–424. Springer, 2020.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Wang *et al.*, 2021] 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*, 44(9):5261–5279, 2021.

[Wang *et al.*, 2023] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Unsupervised Learning of Graph Matching With Mixture of Modes Via Discrepancy Minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–18, 2023.

[Yu *et al.*, 2019] 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*, 2019.

[Zeng and Xie, 2021] Jiaqi Zeng and Pengtao Xie. Contrastive self-supervised learning for graph classification. In

Proceedings of the AAAI conference on Artificial Intelligence, volume 35, pages 10824–10832, 2021.