



ADMatcher: Self-supervised Subgraph Matching via Adaptive Dense-Aware Graph Contrastive Learning

Hao Yan^{1,2}, Lei Du^{1,2}, Mengjiao Wang¹, Xiangyu Song², Yan Jia^{1,2}, Liyi Zeng²,
and Zhaoquan Gu^{1,2}

¹ School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China

{22b951025, 21b951040, 23s151059}@stu.hit.edu.cn

² Department of New Networks, Pengcheng Laboratory, Shenzhen, China


{songxy02, jiay, zengly, guzhq}@pcl.ac.cn

Abstract. Subgraph matching, which aims to determine whether a query graph is a substructure of a target graph, is a crucial problem in various domains. However, subgraph matching is an NP-complete problem. Although prior studies have overcome the computational bottleneck as a supervised learning task, such methods are impractical for label-lacking real scenarios. Therefore, addressing the subgraph matching effectively under limited or without labels is still a key problem. To tackle this problem, we propose ADMatcher, a self-supervised Subgraph Matching method via Adaptive Dense-Aware Graph Contrastive Learning (GCL). We leverage GCL with adaptive augmentation and ensure the augmented graph is a subgraph of the original graph during augmentation stage to learn the subgraph relationship. Specifically, as traditional stochastic augmentations may severely damage the intrinsic properties, we add three dense subgraph augmentation strategies to aware internal useful information. Furthermore, since general data augmentations often use manually selecting strategies, we adopt a min-max optimization method that adaptively adjusts the proportions of different augmentation strategies. Eventually, we redesign the encoder and train the model without labels in self-supervised way. Experiments demonstrate that ADMatcher can outperform the supervised state-of-the-art method with uniformly distributed features, adaptively select better augmentation strategies, and achieve faster training and convergence.

Keywords: Subgraph Matching · Graph Contrastive Learning · Adaptive Augmentation · Dense Aware.

1 Introduction

Graphs are widely used as a powerful data representation tool in various domains because they capture complex relationships and dependencies between objects. Graph matching aims to establish a structural alignment between two input graphs, where a smaller query graph is isomorphic to a subset of a larger target graph. This alignment

 Corresponding authors: Liyi Zeng and Zhaoquan Gu

facilitates profound graph comparisons, analysis, and inference, making it a fundamental element across numerous research disciplines, such as bioinformatics [1], cheminformatics [18], computer security [16], etc.

Research on subgraph matching can be categorized into two primary streams: combinatorial optimization (CO)-based methods [23] and learning-based methods [15]. CO-based methods aim to identify all exact matches. Nevertheless, they incur substantial computational overheads stemming from the NP-complete nature of the problem [7, 22]. Learning-based methods treat it as a supervised learning task, utilizing Graph Neural Networks (GNNs) as encoder. These methods inherently depend on supervised learning, requiring labeled data for training. However, in most realistic contexts, the process of labeling such data frequently proves impractical and costly. Consequently, it becomes essential to investigate self-supervised learning approaches.

Self-supervised GNNs leverage the inherent structure and patterns in unlabeled graph data to learn meaningful representations without requiring clear labels. Thus, contrastive graph learning [27] (GCL) has garnered significant attention and can be used for subgraph matching tasks. Moreover, in GCL, there also exist some issues that need to be addressed. On one hand, random augmentation is usually used in the data augmentation stage [27]. Considering that nodes and edges usually have different importance levels in the graph, stochastic augmentation may destroy the inherent properties of the graph. A better augmentation strategy should be more likely to retain more important components in the original graph. On the other hand, different graph datasets are diverse. For each dataset, augmentation methods must be selected via experience or trial and error. This causes a large consumption of time and computing power.

Overall, to propose a graph contrastive learning subgraph matching network under limited labels, three challenges should be addressed:

1. How to train an efficient subgraph matching network without label?
2. How to retain key properties information in the data augmentation of GCL?
3. How to make the dataset adaptively select data augmentation methods to acquire better experimental results?

To address the above mentioned challenges, we propose **ADMatcher**, A Self Supervised Subgraph Matching via Adaptive Dense-Aware Graph Contrastive Learning. Firstly, to overcome the label scarcity (Challenge 1), our methodology can learn efficacious subgraph representations through self-supervised means. To the best of our knowledge, ours is the pioneer study to embed GCL into subgraph matching and the first to undertake subgraph matching tasks without label introduction during the training stage. Then, to address the destruction of inherent properties in GCL (Challenge 2), we identify the inapplicability of stochastic augmentation and analyze the underlying possible reasons. Subsequently, we perform dense subgraph augmentation, including random walk, k-core, and k-truss subgraph augmentations, to counteract the stochastic augmentation that might damage the intrinsic properties. Meanwhile, to overcome the inability to adaptively and dynamically select augmentation methods (Challenge 3), we devise a min-max optimization method that adaptively adjusts the proportions of different augmentation strategies. This allows each dataset to adaptively select the most suitable data augmentation method, ensuring better experimental results. Finally, we conduct detailed experiments to validate the effectiveness of ADMatcher. We compare

ADMatcher with supervised state-of-the-art (SOTA) methods, explore the effectiveness of adaptive augmentation, and finally perform ablation experiments. Extensive experiments on different synthetic and real-world datasets demonstrate that ADMatcher can achieve SOTA performance comparable to supervised methods. Notably, it shows even better performance when only learning structural information and converges rapidly. We release the source code of ADMatcher ¹ after paper is published.

Our contributions are summarized as follows:

- * We propose ADMatcher by incorporating GCL into the subgraph matching problem. In this way, we develop a promising approach to leverage the power of self-supervised learning without explicit labels.
- * We perform dense subgraph augmentation to overcome stochastic augmentation that may damage the intrinsic properties.
- * We design a min-max optimization method that adaptively and dynamically changes the proportions of six different augmentation strategies.
- * We conduct extensive experiments show that ADMatcher can achieve state-of-the-art performance comparable to supervised methods.

2 Related Work

2.1 Combinatorial Optimization Subgraph Matching Algorithms

CO-based algorithms [23] aims to identify all exact matches. Nevertheless, they incur substantial computational overheads stemming from the NP-complete nature of the problem [7, 22], particularly when handling large-scale graphs. To alleviate the computational cost, recent studies have explored indexing and decompositional strategies [5, 11] or approximate matching to seek inexact solutions [17]. Despite their effectiveness, there is still a need for further exploration in handling larger and more complex graphs.

2.2 Learning Based Subgraph Matching Algorithms

Nowadays, the majority of studies prioritize learning-based methods. Learning-based methods usually calculate the similarity of query and target graphs by comparing embedding vectors. For example, SimGNN [2] first uses GNNs for graph representation learning and a neural tensor network for graph pair matching. NeuroMatch [15] trains a GNN with order embeddings to learn it. These methods get high computational efficiency in graph-level representation but lose node-level details.

2.3 Graph Contrastive Learning

The core idea of contrastive learning is to learn representations that agree with each other under proper transformations, which has led to a surge of interest in visual representation learning [4, 10]. For graph data, most traditional data augmentation strategies in GCL(Graph Contrastive Learning) are randomized. In GraphCL [27], the probabilities of the data augmentation operations, such as removing nodes and edges, are stochastic.

¹ Our source code will be published at <https://github.com/SpringForest24/>

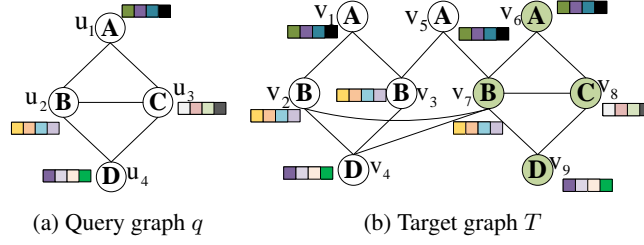


Fig. 1: Subgraph matching problem

3 Problem Formulation and Preliminaries

3.1 Subgraph Matching Problem Statement

Let $G_q = (V_q, E_q)$ be a query graph with associated node features X_q . Meanwhile, $G_T = (V_T, E_T)$ is a large target graph with associated node features X_T . A subgraph matching algorithm aims to confirm whether the query graph G_q is contained in the data graph G_T . When node and edge features are present, the subgraph matching algorithm further requires matching these features. As shown in Fig. 1, Fig. 1a is the query graph, and Fig. 1b is the target graph. The green parts in the target graph $\{v_6, v_7, v_8, v_9\}$ and the query graph $\{u_1, u_2, u_3, u_4\}$ not only have the same structure but also match the node features. Therefore, the query graph G_q can be considered a subgraph of the target graph G_T .

3.2 Graph Contrastive Learning Process

Graph contrastive learning involves a series of crucial steps to learn effective graph representations. It includes data augmentation, encoder, projection head, and loss calculation. Data augmentation gives rise to multiple graph views. For instance, in molecular graphs, operations such as randomly adding or removing nodes/edges are employed. The encoder, typically a GNN, takes augmented graphs, extracts and encodes key features and topology, transforming the data for further processing. Then, a projection head projects encoded representations to enhance discriminability, crucial for distinguishing graph instances in contrastive learning. Finally, the loss is computed. The basic loss function (InfoNCE) [27] for a pair of graphs G_1 and G_2 with representations z_1 and z_2 is:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(z_1, z_2)/\tau)}{\sum_{i,j} \exp(\text{sim}(z_i, z_j)/\tau)}$$

where τ is a temperature parameter, sim is cosine similarity. For similar graph pairs (G_1, G_2) , z_1 and z_2 should be close, so the similarity is large and the loss is small.

4 The Proposed Method

In this section, we introduce ADMatcher in detail. The overall framework is shown in Fig. 2. The upper part is the ADMatcher training stage, and the lower part is the

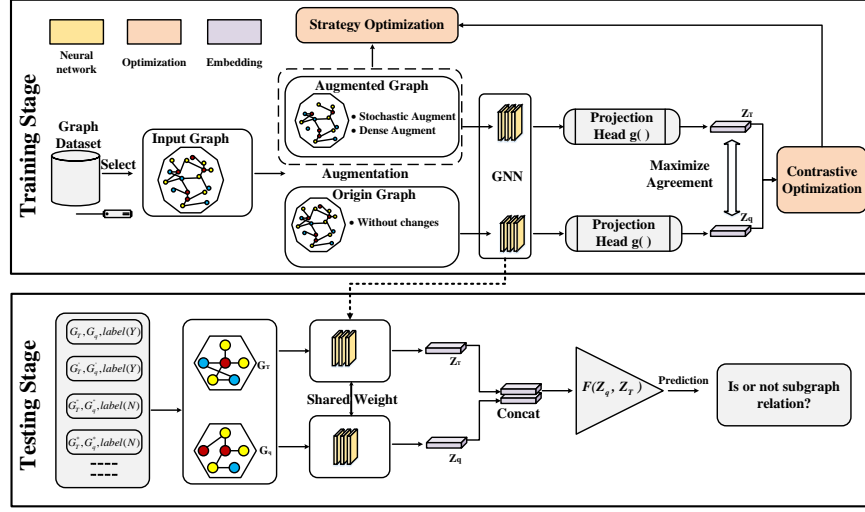


Fig. 2: Overview of ADMatcher framework.

testing stage. In the training stage, we first select any graph dataset (with or without labels). Then, we perform stochastic and dense augmentation methods for each graph and input them into the GNNs network $f(\cdot)$. After a projection head $g(\cdot)$, we get the embedding vectors z_i and z_j of the two graphs. Finally, it is performed by maximizing the agreement between the origin graph and the augmentation subgraph in the embedding space. Furthermore, we design a min-max optimization method to allow the model to determine the appropriate combination of augmentation methods among the six graph augmentation methods through its own optimization process. Through the contrastive optimization module and the strategy optimization module, we finally select the appropriate augmentation methods. In the testing stage, we manually label some graph pairs for subgraphs and non-subgraph relations. Their format is (G_T, G_q, label) . Then, we use the trained GNN to compute the embedding of G_T and G_q after concatenating the two embeddings. Finally, we train a linear SVM classifier based on graph embeddings for classification (is or is not subgraph relation).

4.1 Subgraph Augmentation methods

Data augmentation is a significant part of contrastive learning. However, data augmentation is not task-specific, because contrastive learning can be considered as a pre-trained and fine-tuned model that relies on fine-tuned labels in downstream tasks. Therefore, how to design an augmentation strategy to ensure that the goal of contrastive learning is consistent with the goal of subgraph matching?

The answer is that during the data augmentation process, one of the graphs remains unchanged and the other graph is a subgraph of the original graph to solve this problem. The proof is as follows:

Theorem 4.1: Suppose $A(G)$ is a subgraph of G . As long as the graph neural network f is trained according to the Graph Contrastive Learning (GCL) objective function, can satisfy the subgraph matching judgment function.

PROOF: Let $G = (V, E, X)$ be the original graph and $A(G) = (V', E', X')$ with $V' \in V$ and $E' \in E$. In GCL, consider the contrastive loss \mathcal{L} (in a simplified form similar to InfoNCE [27] for two augmented views).

$$\mathcal{L} = \mathbb{E}_{G \sim \mathbb{P}_G} \left[-\log \left(\frac{\exp(s(f(G), f(A(G))))}{\sum_{G' \sim \mathbb{P}_G} \exp(s(f(G), f(A(G'))))} \right) \right]$$

By maximize the mutual information I , the model endeavors to minimize \mathcal{L} . This, consequently, leads to an increase in $s(f(G), f(A(G)))$. Numerous subgraph relationship determination functions bear resemblance to $s(f(G), f(A(G)))$, relying on the assessment of similar distances. In summary, as long as the enhanced graph is kept as a subgraph of another graph, the contrastive learning training goal and the subgraph matching training goal can be made consistent.

Stochastic Augmentation. Stochastic Augmentation is a commonly used augmentation method in GCL. In this part, the following methods are used:

Node Drop and Edge Drop: We set the probability P_{drop} of node-dropping or edge-dropping operations on the augmented graph of the original graph. Each node and edge in the graph has the same probability of keeping, $P_{keep} = 1 - P_{drop}$. Then, we remove nodes and edges randomly using the Bernoulli distribution; **Feature Mask** does not change the graph structure, encouraging the model to learn with fewer node features and more structure information. We also set the probability P_n of the node feature mask. Afterward, these node features are masked by the normal distribution.

Dense Subgraph Augmentation. Common topology augmentation strategies include node dropping, edge removal, etc. These cannot retain important information in the graph. A better augmentation strategy should more likely retain the more important components of the original graph.

Random Walk Sample samples a subgraph from G using random walk. The underlying assumption is that the semantics of G can be largely preserved in its (partial) local structure; **K-core and K-truss subgraph:** K-core [20] ensures that every node in the subgraph has at least k neighbors, represented as $deg(v) \leq k$, maintaining essential connectivity. K-truss [6] guarantees that each edge is part of at least $(k - 2)$ triangles, capturing local clustering structures.

4.2 Adaptive Change Proportions of Augmentations

Since we proposed six subgraph augmentation methods, each augmentation method consumes a lot of time and computing power. In addition, since each dataset is manually selected separately, it is difficult to directly apply a successful augmentation method selection strategy to other datasets. Therefore, we get the idea from [26] and align with the idea of adversarial training, adopting a min-max optimization method that adaptively and dynamically adjusts the proportions for six augmentation strategies.

Double-layer Optimization Framework. ADMatcher uses a double-layer optimization framework, namely contrastive optimization and strategy optimization. In contrastive optimization, the commonly used InfoNCE is used, and in strategy optimization, the sampling distribution of the augmentation method is jointly optimized. Specifically, it is achieved through the following double-layer optimization problem:

$$\begin{aligned} & \min_{\theta} \mathcal{L}(G, A_1, A_2, A_3, A_4, A_5, A_6, \theta), \\ & \text{s.t. } \mathbb{P}(A_1, A_2, A_3, A_4, A_5, A_6) \in \arg \min_{\mathbb{P}(A)} \mathcal{D}(G, A'_1, A'_2, A'_3, A'_4, A'_5, A'_6, \theta) \end{aligned}$$

where θ is the model parameter, \mathcal{L} is the contrastive loss function, G is the input graph, and A_n is an augmentation method sampled from stochastic and dense augmentation. \mathcal{D} is the strategy optimization to adjust the sampling distribution \mathbb{P} .

Min-Max Optimization Method. Inspired by adversarial training, the ADMatcher framework is instantiated into a minimum-maximum optimization form:

$$\begin{aligned} & \min_{\theta} \mathcal{L}(G, A_1, A_2, A_3, A_4, A_5, A_6, \theta), \\ & \text{s.t. } \mathbb{P} \in \arg \max_{\mathbb{P}} \left\{ \mathcal{L}(G, A'_1, A'_2, A'_3, A'_4, A'_5, A'_6, \theta) - \frac{\gamma}{2} \text{dist}(\mathbb{P}, \mathbb{P}_{prior}) \right\} \end{aligned}$$

where $\gamma \geq 0$ and \mathbb{P}_{prior} are the prior distributions of all possible augmentation methods. dist is the distance function between the sampling distribution and the prior distribution. The idea of adversarial training helps to improve the generalization ability, robustness and transferability of the model.

During the training process, the alternating gradient descent algorithm (AGD) is used to optimize the above problem, alternating upper layer (contrastive optimization) minimization and lower layer (strategy optimization) maximization operations in each iteration.

4.3 Subgraph Contrastive Learning

In this section, we introduce other core parts and the overall process of the ADMatcher framework via providing pseudocode.

GNN-based Encoder. While the subgraph augmentation part has ensured that the augmented view would be subgraphs, the encoder part (graph neural network) may still lose this substructure information during the graph learning process. In general, conventional GNNs follow a message-passing neural network (MPNN) framework, as local information is aggregated and passed to neighbors.

$$a_u^{(l)} = \text{AGG}^{(l)} \left(\{h_u^{(l-1)} : u \in \mathcal{N}(v)\} \right)$$

where $a_u^{(l)}$ represents the aggregated information of node v at layer l . $\mathcal{N}(v)$ denotes the neighborhood set of node v . $\text{AGG}^{(l)}$ is the function used to aggregate neighbor information

$$h_v^{(l+1)} = \text{MLP}^{(l)}\{(1 + \epsilon^{(l)}) \cdot h_v^{(l)} + \text{AGG}^{(l)}(a_u^{(l)})\}$$

We choose Graph Isomorphism Network (GIN) [24] as the encoder of ADMatcher. Then, in the GIN network, $\epsilon^{(l)}$ is a learnable parameter controlling the contribution of the node's representation. $h_v^{(l)}$ stands for the representation of node v at layer l . $\text{MLP}^{(l)}$ denotes the multi-layer perceptron function at layer l . Compared to GCN (Graph Convolutional Network), GIN offers superior expressive power, better generalization, higher robustness, and stronger theoretical guarantees.

Projection Head. The augmented representations are subjected to a non-linear transformation, known as the projection head and denoted as $g()$, which maps them into a distinct latent space. The computation of the contrastive loss takes place within this latent space, following the recommendation provided in the referenced work [4]. In the context of graph contrastive learning, a two-layer Multilayer Perceptron (MLP) is employed to obtain the variables z_i and z_j .

Contrastive Loss. To promote the consistency between positive pairs (z_i, z_j) and differentiate them from negative pairs. In our approach, we employ the InfoNCE loss [27]. During GNN pre-training, a minibatch of N graphs is randomly selected and subjected to contrastive learning. This process generates $2N$ augmented graphs and their corresponding contrastive loss, denoted as $z_{n,i}$ and $z_{n,j}$ for the n th graph in the minibatch. Negative pairs are explicitly sampled from the other augmented graphs within the same minibatch.

SVM Classification. In the testing stage, we label some triplets (G_T, G_q, label) and put these graphs into the GNN encoder to obtain the corresponding embedding vectors. We adopt the evaluation methodology from [28], a linear SVM classifier is trained using graph embeddings for the task of graph classification. Subsequently, this classifier is transferred to the subgraph matching domain. The embeddings of the data graph and the query graph are combined by concatenation and then fed into the SVM classifier. Can linear SVM classification be used in subgraph matching problems? We prove this as follows:

Theorem 4.2: Assuming sufficient training and data augmentation satisfying Theorem 4.1, the embeddings of subgraph relations and non-subgraph relations are linearly separable.

PROOF: Let X_1 denote the set of vectors formed by splicing two vectors corresponding to all subgraph relationships, and X_2 denote the set of vectors formed by splicing two vectors corresponding to all non-subgraph relationships.

We define a distance metric function $d(x)$. Due to adequate training, for any $x_1 \in X_1$, there exists $r_1 > 0$ such that: $d(x_1) \leq r_1$. And for any $x_2 \in X_2$, there must exists

Algorithm 1 ADMatcher: A Self-supervised Sub-Graph Matching via Adaptive Dense-Aware Graph Contrastive Learning

Input: Graph Dataset $\{G_m : m \in M\}$

Parameter: G_m , Encoder $f()$, Projection Head $g()$, temperature parameter τ , prior distribution \mathbb{P}_{prior} , parameter γ

- 1: Initialize $\mathbb{P}(A_1, A_2, A_3, A_4, A_5, A_6)$ with equal proportion
 - 2: Sample minibatch $\{G_n : n \in N\}, G_n \subseteq G_m$
 - 3: **while** $n = 1$ to N **do**
 - 4: Sample $A_1, A_2, A_3, A_4, A_5, A_6$ from $\mathbb{P}(A_1, A_2, A_3, A_4, A_5, A_6)$
 - 5: $G_{n,i} \leftarrow G_n, h_{n,i} \leftarrow f(G_{n,i}), z_{n,i} \leftarrow g(h_{n,i})$
 - 6: $G_{n,j} \sim A(G_n)$ using one of the sampled $A_n, h_{n,j} \leftarrow f(G_{n,j}), z_{n,j} \leftarrow g(h_{n,j})$
 - 7: Compute $\ell_n = -\log \frac{\exp(\sin(z_{n,i}, z_{n,j})/\tau)}{\sum_{n'=1, n' \neq n} \exp(\sin(z_{n,i}, z_{n',j})/\tau)}$
 - 8: $\mathbb{P}(A_n) \leftarrow \text{AdaptiveDoubleLayerOptimization}(\ell, \theta, \mathbb{P}(A_1, A_2, A_3, A_4, A_5, A_6), \mathbb{P}_{prior}, \gamma)$
 - 9: **end while**
 - 10: Sample some triplets (G_T, G_q, label) , using Encoder $f()$ and Projection Head $g()$ to compute the embedding of G_T and G_q
 - 11: Using SVM classification to compute the accuracy
 - 12: **return** Accuracy
-

$r_1 \leq r_2$ such that: $d(x_2) \geq r_2$. Then, we define a hyperplane H in the vector space, a normal vector \mathbf{n} and a bias term b , making $H(X) = \mathbf{n} * f(x) + b$.

$$r_1 \leq r_2 \rightarrow d(x_1) \leq d(x_2) \rightarrow \mathbf{n} * f(x_1) + b \leq \mathbf{n} * f(x_2) + b \rightarrow H(X_1) < H(X_2)$$

Since we can find a hyperplane H to separate subgraph relationships embedding and non-subgraph relationships embedding, linear SVM classification can be used in subgraph matching problems. To ensure the reliability and stability of experimental results, we adopt a 10-fold cross-validation methodology.

Overall Process of ADMatcher. The Alg. 1 shows the the overall Process of ADMatcher. It begins by taking a graph dataset $\{G_m : m \in M\}$ as input and several parameters like encoder $f()$, projection head $g()$, temperature τ , prior distribution \mathbb{P}_{prior} and γ . Firstly, it initializes the sampling distribution $\mathbb{P}(A_1, A_2, A_3, A_4, A_5, A_6)$ equally. Then, a minibatch of graphs is sampled. In the loop, it samples augmentation operations A_n , processes the original and transformed graphs through the encoder $f()$ and projection head $g()$ to obtain representations and calculate the contrastive loss ℓ_n . The sampling distribution $\mathbb{P}(A_n)$ is updated using the **AdaptiveDoubleLayerOptimization** function. After the loop, triplets are sampled and their embeddings are computed. Finally, SVM classification is applied to compute the accuracy. Overall, Alg. 1 outlines a process from data sampling and graph augmentation to loss calculation, distribution update, and accuracy evaluation for self-supervised subgraph matching.

5 Experiments

In this section, we conduct comprehensive experiments to answer the following research questions:

- RQ1: How does our ADMatcher perform compare to SOTA method?
- RQ2: Can augmentation strategies enhance subgraph matching representation?
- RQ3: What is the effect of ADMatcher on hyperparameters?

5.1 Experiment Settings

Datasets. For the synthetic dataset, we use PyG [8] to generate a dataset by utilizing ER-random graphs and WS-random graphs [15]. It contains 1,000 graphs, each with an average of 30 nodes. For the real-world datasets, we choose two biomedical graph datasets ENZYMES, PROTEINS [3], and three chemical graph datasets MUTAG [12], cox2 [21] and AIDS [19]. Moreover, we choose a social graph dataset, IMDB-BINARY [25]. Additionally, the Synthetic dataset is a complex dataset. IMDB-BINARY has the highest average node degree, which is 9.8. The other datasets are below 5.

Baseline. We select four comparison methods. **GMN** [13] and **SimGNN** [2] are widely used for computing the similarity between two graphs. To adapt them for subgraph matching, we reformulate the training objective and output layer to better suit the subgraph matching problem. Both of them are learning-based methods. **NeuroMatch** [15] is also a learning-based method that decomposes the query and target graphs into smaller subgraphs and employs the order embedding approach for subgraph matching. In contrast, **D2Match** [14] transforms the subgraph matching problem into checking whether the corresponding subtrees rooted at these two nodes are subtree isomorphic. Yet, D2Match is not a learning-based method.

Implementation Details. The graph representation module is implemented via PyTorch, PyG [8], Scikit-learn and NetworkX [9]. We adopt the commonly used unsupervised representation learning setting for GCL benchmarks [28]. In the testing stage, we employ these raw graphs as target graphs and generate the query graphs using breadth-first search sampling from the target graphs as triplets. For non-subgraph relation triples, we select the target graph and query graph from different datasets to ensure the true non-subgraph relation. We split triplets into training and testing at a ratio of 4 : 1 and report the average classification accuracy under SVM classification for 10-fold cross-validation and repeating each experiment five times. Experiments are run on a server with an Intel CPU and NVIDIA 3090 GPU. The operating system is Linux(CentOS).

5.2 Comparison with SOTA Methods (RQ1)

Table. 1 shows the overall performance of all compared models. There are four algorithms for comparison: GMN [13], SimGNN [2], NeuroMatch [15] and D2Match[14]. ADMatcher reaches the performance levels of state-of-the-art supervised methods across

Table 1: Overall performance comparison in terms of accuracy

	Synthetic	ENZYMES	PROTEINS	IMDB-BINARY	MUTAG	COX2	AIDS
GMN[13]	56.6±8.61	89.4±16.44	93.8±2.41	69.3±15.18	90.8±6.16	69.7±18.20	78.3±6.92
SimGNN[2]	70.5±2.72	98.6±1.08	96.2±0.97	85.0±19.58	98.7±0.60	99.9±0.22	96.5±0.68
NeuroMatch[15]	65.7±8.98	97.9±1.08	94.5±1.73	86.5±6.51	99.2±0.22	100.0±0.00	97.4±0.96
D2Match[14]	74.3±0.22	99.9±0.22	100.0±0.00	93.3±1.03	100.0±0.00	100.00±0.00	99.5±0.27
ADMatcher	88.49±1.26	96.83±0.51	95.66±0.35	96.75±0.63	97.31±0.41	99.14±0.55	95.85±0.60
ADMatcher(adaptive)	86.34±3.25	94.03±0.86	93.10±0.11	97.67±0.66	93.33±1.18	98.58±0.26	94.21±1.00

Table 2: Comparing with uniform distribution feature in terms of accuracy

	Synthetic	ENZYMES*	PROTEINS*	IMDB-BINARY*	MUTAG*	COX2*	AIDS*
SimGNN[2]	84.1±3.40	76.0±2.12	64.6±3.36	72.0±2.45	80.3±6.18	88.2±2.05	73.2±6.06
NeuroMatch[15]	74.5±2.57	86.6±3.64	52.8±4.76	60.4±10.11	90.4±2.88	91.0±5.70	75.6±17.78
D2Match[14]	86.6±1.44	96.0±2.16	83.4±2.97	90.2±1.79	99.2±0.84	99.8±0.45	95.0±1.41
ADMatcher	88.24±1.36	96.67±0.82	95.46±0.23	89.24±1.32	96.71±1.36	100.0±0.00	95.65±0.06
ADMatcher(adaptive)	84.79±1.09	95.56±0.39	94.59±0.96	85.08±14.03	95.42±2.71	100.00±0.00	94.83±0.42

multiple datasets. Specifically, it attains the highest performance on the synthetic dataset with an improvement of over 10%, and also performs outstandingly on IMDB - BINARY. Although it does not perform as well as D2Match on the ENZYMES, PROTEINS, MUTAG, and AIDS datasets. D2Match only exceeding our ADMatcher and NeuroMatch methods by 1 - 2 percentage points in accuracy. It does not definitively prove D2Match’s superiority, and further experiments under additional conditions are required for a more conclusive comparison.

To further evaluate the impact of NeuroMatch, D2Match, and ADMatcher, we remove the node features and edge features from the graph dataset and assign uniform distribution features to the nodes and edges. GMN performs poorly in subgraph matching, so we focus on comparing NeuroMatch, D2Match, and ADMatcher. In Table. 2, * represents datasets with uniform distribution. The performance of SimGNN and NeuroMatch lags behind D2Match and ADMatcher. D2Match underperforms ADMatcher on most datasets. Despite D2Match using complexity for accuracy in subtree conversion, ADMatcher outperforms it with uniform node distribution, showing its graph topology capture and generalization ability.

Meanwhile, we compare the running time. ADMatcher needs 200 epochs which has much lower the running time than NeuroMatch (100,000 epochs) and D2Match

Table 3: Single epoch runtime comparison

	Synthetic	ENZYMES	PROTEINS	IMDB-BINARY	MUTAG	COX2	AIDS
NeuroMatch[15]	3.87	1.14	3.10	3.46	0.55	1.42	1.96
D2Match[14]	4.41	1.26	3.21	3.79	0.68	2.07	2.33
ADMatcher(one)	0.49	0.14	0.36	0.33	0.06	0.10	1.02
ADMatcher(adaptive)	7.46	1.88	5.03	5.28	0.29	1.26	3.41

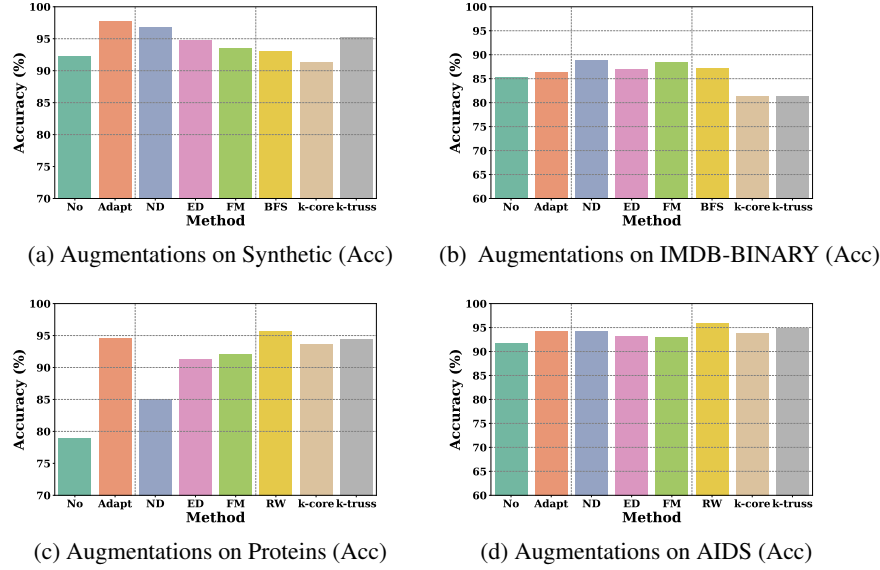


Fig. 3: Accuracy of different augmentations on different datasets

(500 epochs). To maintain fairness, we compare the iteration time of a single epoch in Fig. 3. The single enhanced ADMatcher takes less time per iteration than the comparison methods, though the time per epoch in the Adaptive multiple augmentation method of ADMatcher is longer. So, with the right augmentation method, ADMatcher can be far superior in running time to the comparison algorithms.

5.3 The Role of Subgraph Augmentations (RQ2)

Through experiment 5.2 results methods above, The accuracy of ADMatcher (adaptive) is not consistently superior to that of the single-augmentation ADMatcher. Additionally, the running time of ADMatcher (adaptive) is also longer.

Nevertheless, the ADMatcher (adaptive) method has the advantage of choosing the augmentation method that is most suitable for the dataset. After using the adaptive ADMatcher for a few short iterations, a suitable augmentation method can be found, and then using a single augmentation method can greatly reduce the time and obtain a higher accuracy.

Table. 4 shows the percentage of different augmentation methods selected by Adaptive ADMatcher. ND, ED, FM, respectively represent node drop, edge drop and feature mask in stochastic augmentation. RW, K-Core and K-truss respectively represent random walk sample, k-core and k-truss in Dense subgraph Augmentation. We can see that for the Synthetic dataset, the preference for random augmentation and dense subgraph augmentation is basically the same, and random augmentation is only 1% higher than dense subgraph augmentation. For the densest graph dataset IMDB-BINARY, ran-

Table 4: Proportion of different augmentations

	ND	ED	FM	RW	K-Core	K-Truss
Synthetic	50.99%	0.00%	0.00%	0.00%	0.00%	49.00%
ENZYMES	0.05%	0.00%	0.00%	79.32%	0.00%	20.63%
PROTEINS	0.00%	0.00%	0.00%	76.34%	0.00%	23.66%
IMDB-BINARY	84.30%	0.00%	0.00%	0.00%	0.00%	15.70%
MUTAG	4.81%	0.01%	0.00%	94.55%	0.63%	0.00%
COX2	2.36%	0.68%	0.00%	94.76%	1.40%	0.80%
AIDS	1.48%	10.77%	0.00%	85.82%	1.58%	0.36%

dom augmentation (node drop) is preferred. For other biomedical datasets and chemical datasets, dense subgraph augmentation (random walk and K-truss) is preferred.

To verify whether the adaptive ADMatcher select the correct data augmentation methods, we compare the accuracies under different augmentation effects for the same datasets. We select dataset respectively from artificial datasets (Synthetic), social datasets (IMDB-BINARY), biological datasets (PROTEINS), and chemical datasets (AIDS) for experiment. It can be seen from Fig. 3 that subgraph augmentations play a crucial role in graph contrastive learning for the subgraph matching problem. Without adopting augmentations, the overall accuracy decreases. Meanwhile, if inappropriate augmentation methods are utilized, the accuracy may also decline.

The appropriate augmentation methods shown in Table. 4 can all exhibit excellent performance in Fig. 3. For example, for Proteins and AIDS in Table. 4, they are more suitable for the Random walk and K-truss augmentations among the dense subgraph augmentations. Consequently, the Random walk and K-truss augmentations demonstrate high accuracy in Fig. 3a and Fig. 3b. The dense subgraph augmentation operation can better maintain the integrity and relevance of these internal links when augmenting the data, thereby helping the model learn the key features and patterns in the biomedical and chemical dataset. Besides, for complex and high avg. node degree datasets, stochastic augmentation is better. Messy synthetic datasets lack regular internal structs. Random node/edge deletion breaks chaos, making model explore diverse local information.

5.4 Ablation Study (RQ3)

In this section, we perform ablation studies for GNN embedding model, probability of augmentation, stochastic vs. dense-subgraph augmentation and carry out convergence analysis.

Firstly, we test the effect of layers of the encoder and changed them from 1 to 5. We select four datasets as experiment 5.3. Results in Fig. 4a show that ADMatcher can achieve better results when there are few encoder layers, and the accuracy does not significantly improve as the number of encoder layers increases. Therefore, we choose a 2-layer encoder to ensure accuracy and reduce computational complexity.

Then, we test the effect of the probability of augmentation on the experimental results, with the augmentation ratio of 20%, 50% and 80% in Fig. 4b. For complex

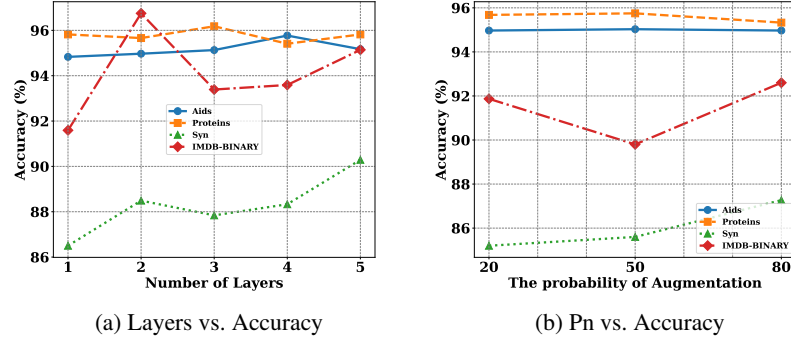


Fig. 4: The influence of hyperparameters on experimental results

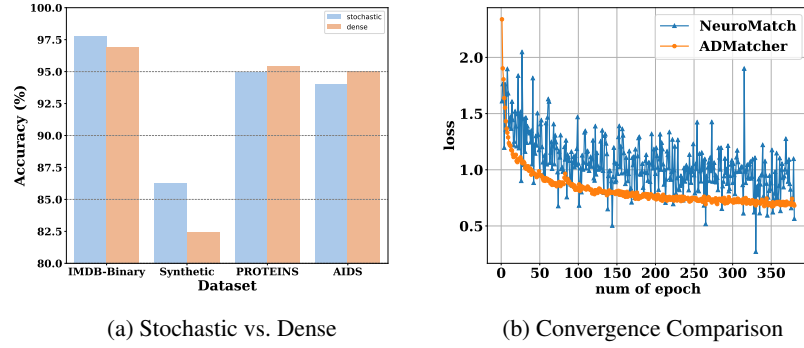


Fig. 5: Random vs Dense Subgraph Augmentation and convergence analysis

datasets and datasets with higher average node degrees, changing the augmentation probability affect the results, but for biological and chemical datasets, the change is not significant. We believe the key lies in the carefully designed and optimized data augmentation, encoding and loss minimization components. These aspects are interdependent and must be optimized in coordination, rather than simply increasing the complexity of the encoding part or increasing the ratio of data augmentation.

Following, in order to test the impact of augmentation categories on different datasets on the experimental results in Fig. 5a. We conduct experiments using both random augmentation and dense subgraph augmentation. The final results show that IMDB-BINARY and Synthetic perform better under random augmentation, while biological and chemical datasets perform better on dense subgraph augmentation.

Finally, we analyze convergence. As D2Match isn't learning-based and can't compare loss changes, we compare ADMatcher with NeuroMatch, the second-best learning-based method. Fig. 5b shows their training loss on ENZYMES. For fairness, ADMatcher's loss function is reduced ten times due to different loss functions. Using the

Enzymes dataset and setting 400 iteration epochs, ADMatcher converges faster via contrastive learning and handling multiple samples. Moreover, compared to ADMatcher, NeuralMatch has larger loss fluctuation and poorer stability.

6 Conclusion

In this paper, we present ADMatcher, a novel self-supervised approach for subgraph matching that incorporates adaptive dense-aware graph contrastive learning. By integrating dense subgraph augmentation strategies and adaptive adjustment augmentation methods, ADMatcher effectively addresses the subgraph matching problem under limited labels. Experimental results validate that ADMatcher can adaptively choose augmentation methods and rapidly attain state-of-the-art performance comparable to supervised methods. Overall, ADMatcher enriches the subgraph matching toolkit via adaptive dense-aware GCL and unlocks new prospects for processing real-world graph data lacking sufficient labels.

Acknowledgments

This work was supported in part by the Shenzhen Science and Technology Program (No. KJZD20231023094701003), the Major Key Project of PCL (Grant No. PCL2024A05), and the National Natural Science Foundation of China (Grant No. 62372137).

References

1. Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., Sahinalp, S.C.: Biomolecular network motif counting and discovery by color coding. *Bioinformatics* **24**(13), i241–i249 (2008)
2. Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., Wang, W.: Simgnn: A neural network approach to fast graph similarity computation. In: *Proceedings of the twelfth ACM international conference on web search and data mining*. pp. 384–392 (2019)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. *Bioinformatics* **21**, i47–i56 (2005)
4. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: *International conference on machine learning*. pp. 1597–1607 (2020)
5. Choudhury, S., Holder, L., Chin, G., Agarwal, K., Feo, J.: A selectivity based approach to continuous pattern detection in streaming graphs. *arXiv preprint arXiv:1503.00849* (2015)
6. Cohen, J.: Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* **16**, 1–29 (2008)
7. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* **26**(10), 1367–1372 (2004)
8. Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428* (2019)
9. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. *Tech. rep.*, Los Alamos National Lab., Los Alamos, NM (2008)

10. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9729–9738 (2020)
11. Kim, K., Seo, I., Han, W.S., Lee, J.H., Hong, S., Chafi, H., Shin, H., Jeong, G.: Turboflux: A fast continuous subgraph matching system for streaming graph data. In: Proceedings of the 2018 international conference on management of data. pp. 411–426 (2018)
12. Kriege, N., Mutzel, P.: Subgraph matching kernels for attributed graphs. In: Proceedings of the 29th International Conference on Machine Learning. pp. 291–298 (2012)
13. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: International conference on machine learning. pp. 3835–3845 (2019)
14. Liu, X., Zhang, L., Sun, J., Yang, Y., Yang, H.: D2match: leveraging deep learning and degeneracy for subgraph matching. In: International Conference on Machine Learning. pp. 22454–22472 (2023)
15. Lou, Z., You, J., Wen, C., Canedo, A., Leskovec, J., et al.: Neural subgraph matching. arXiv preprint arXiv:2007.03092 (2020)
16. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrishnan, V.: Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 1795–1812 (2019)
17. Mongiovi, M., Di Natale, R., Giugno, R., Pulvirenti, A., Ferro, A., Sharan, R.: Sigma: a set-cover-based inexact graph matching algorithm. *Journal of bioinformatics and computational biology* **8**(02), 199–218 (2010)
18. Raymond, J.W., Gardiner, E.J., Willett, P.: Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *Journal of chemical information and computer sciences* **42**(2), 305–316 (2002)
19. Riesen, K., Bunke, H.: Iam graph database repository for graph based pattern recognition and machine learning. In: Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop. pp. 287–297. Springer (2008)
20. Seidman, S.B.: Network structure and minimum degree. *Social networks* **5**(3), 269–287 (1983)
21. Sutherland, J.J., O’Brien, L.A., Weaver, D.F.: Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences* **43**(6), 1906–1915 (2003)
22. Ullmann, J.R.: An algorithm for subgraph isomorphism. *Journal of the ACM* **23**(1), 31–42 (1976)
23. Vesselinova, N., Steinert, R., Perez-Ramirez, D.F., Boman, M.: Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access* **8**, 120388–120416 (2020)
24. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)
25. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1365–1374 (2015)
26. You, Y., Chen, T., Shen, Y., Wang, Z.: Graph contrastive learning automated. In: International Conference on Machine Learning. pp. 12121–12132 (2021)
27. You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., Shen, Y.: Graph contrastive learning with augmentations. *Advances in neural information processing systems* **33**, 5812–5823 (2020)
28. Zhu, Y., Xu, Y., Liu, Q., Wu, S.: An empirical study of graph contrastive learning. arXiv preprint arXiv:2109.01116 (2021)