D2Match: Leveraging Deep Learning and Degeneracy for Subgraph Matching

Xuanzhou Liu [1 2 †] Lin Zhang [2] Jiaqi Sun [1] Yujiu Yang [1 *] Haiqin Yang [2 *]

Abstract

Subgraph matching is a fundamental building block for graph-based applications and is challenging due to its high-order combinatorial nature. Existing studies usually tackle it by combinatorial optimization or learning-based methods. However, they suffer from exponential computational costs or searching the matching without theoretical guarantees. In this paper, we develop D^2Match by leveraging the efficiency of Deep learning and Degeneracy for subgraph matching. More specifically, we first prove that subgraph matching can degenerate to subtree matching, and subsequently is equivalent to finding a perfect matching on a bipartite graph. We can then yield an implementation of linear time complexity by the built-in tree-structured aggregation mechanism on graph neural networks. Moreover, circle structures and node attributes can be easily incorporated in D^2Match to boost the matching performance. Finally, we conduct extensive experiments to show the superior performance of our D^2Match and confirm that our D^2Match indeed exploits the subtrees and differs from existing GNNs-based subgraph matching methods that depend on memorizing the data distribution divergence.

1. Introduction

Subgraph isomorphism, or subgraph matching at the node level (McCreesh et al., 2018), is a critical yet particularly challenging graph-related task. It aims to determine whether a query graph is isomorphic to a subgraph of a large target graph. It is an essential building block for various graph-based scenarios, e.g., alignment of cross-domain data (Chen et al., 2020), temporal alignment of time series (Zhou & Torre, 2009), and motif matching (Milo et al., 2002; Peng et al., 2020), etc.

Existing studies for subgraph matching can be divided into two main streams: combinatorial optimization (CO)-based and learning-based methods (Vesselinova et al., 2020). Early algorithms often formulate subgraph matching as a CO problem that aims to find all exact matches in a target graph. Unfortunately, this leads to an NP-complete issue (Ullmann, 1976; Cordella et al., 2004), which suffers from exponential time costs. To alleviate the computational cost, researchers have employed approximate techniques to seek inexact solutions (Mongiovì et al., 2010; Yan et al., 2005; Shang et al., 2008), which yield suboptimal matchings. An alternative solution is to frame subgraph matching as a supervised learning problem (Bai et al., 2019; Rex et al., 2020; Bai et al., 2020), which utilizes the Graph Neural Networks (GNNs). However, the learning-based or GNN-based methods mainly aim to optimize the representations in a black-box way. The lack of theoretical guarantees makes them inexplicable and often cannot seek the exact match subgraphs.

In order to tackle the above challenges, we propose a white-box GNN-based solution, D^2Match, to leverage the efficiency of Deep GNNs and Degeneracy for subgraph matching. With rigorous theoretical proofs, we provide explainable results at each learning step. We first prove that finding the matched nodes between the query graph and the target one can degenerate to check whether the corresponding subtrees rooted at these two nodes are subtree isomorphic. This degeneration allows us to check whether a perfect matching exists on a bipartite graph, which results in a polynomial time complexity solution. The above two steps convert subgraph matching into computing an indicator matrix whose elements represent the subtree isomorphic relationship between nodes in the query graph and the target one. Hence, the matching matrix required by the CO-based methods for subgraph matching degenerates to seeking an indicator matrix, which is computed by GNNs via its intrinsic tree-structured aggregation mechanism.

Note that this implementation battles the matching mechanism directly by GNNs rather than optimizing the representations of GNNs as in the existing work. Our implementation allows us to reduce the time cost of perfect matching

[1]Shenzhen International Graduate School, Tsinghua University, Shenzhen, China [2]International Digital Economy Academy (IDEA). †Work done when Xuanzhou was interned at IDEA. Correspondence to: Yujiu Yang <yang.yujiu@sz.tsinghua.edu.cn>, Haiqin Yang <hqyang@ieee.org>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

from polynomial time to linear time. Moreover, we can easily incorporate other information, including circle structures (abstracted as *supernodes*) and node attributes, into our D^2Match to boost the matching performance.

Our contribution is four-fold: (1) We propose the first white-box GNN-based model, called D^2Match, to leverage deep learning and degeneracy for subgraph matching. We provide rigorous theoretical proofs to guarantee that subgraph matching can degenerate to subtree matching, and finally perfect matching on a bipartite graph. (2) To the best of our knowledge, this is the first GNN-based model to tackle subgraph matching directly, which degenerates the matching matrix required by the CO-based methods to an indicator matrix computed by GNNs via the intrinsic tree-structured aggregation mechanism. This allows us to compute the indicator matrix in linear time. (3) Our D^2Match can easily include other information, including circle structures and node attributes to boost the model performance. (4) Extensive empirical evaluations show that D^2Match outperforms state-of-the-art subgraph matching methods by a substantial margin, and uncovered that learning-based methods tend to capture the divergence of the data distribution rather than exploiting graph structures.

2. Related work

Subgraph matching is to check whether a query graph is subgraph isomorphic to the target one (McCreesh et al., 2018). Here, we highlight three main lines of related work:

Combinatorial optimization (CO)-based methods first tackle subgraph matching by only modeling graph structure (Ullmann, 1976). Some later work starts to facilitate both graph structure and node attributes (He & Singh, 2008; Shang et al., 2008; Han et al., 2013; Bhattarai et al., 2019). These combinatorial optimization methods often rely on backtracking (Priestley & Ward, 1994), i.e., heuristically performing matching on each pair of nodes from the query and the target graphs. Such methods suffer from exponential computing costs. A mitigated solution is to employ an inexact matching strategy. Early methods first define metrics to measure the similarity between the query graph and the target graph. Successive algorithms follow this strategy and propose more complex metrics. For example, Mongiovì et al. (2010) convert the graph matching problem into a set-cover problem to attain a polynomial complexity solution. Yan et al. (2005) introduce a thresholding method to filter out mismatched graphs. Khan et al. (2011) define a metric based on neighborhood similarity and employ an information propagation model to find similar graphs. Kosinov & Caelli (2002) and Caelli & Kosinov (2004) align the nodes' eigenspace and project them to the eigenspace via clustering for matching. However, most of these algorithms cannot scale to large graphs due to the high computational cost, and their hand-crafted features make them hard to generalize to complex tasks.

Learning-based methods typically compute the similarity between the query and target graphs, e.g., comparing their embedding vectors. Bai et al. (2019) adopt GNNs to learn representations of the graphs and employs a neural tensor network to match the representation of graph pairs. One immediate challenge is that a single graph embedding vector cannot capture the partial order of subgraph isomorphism. Thus, Rex et al. (2020) train a GNN model to represent graphs while incorporating order embeddings to learn the partial order. These methods can compute graph-level representations, achieving high computational efficiency. However, they miss the node-level information, which may lose critical details in subgraph matching. To perform node-level matching, several methods (Bai et al., 2020; Li et al., 2019) introduce the node-level representation into the problem. These methods often adopt different attention mechanisms to generate pairwise relations. However, abusing the attention mechanism makes the model lack interpretability and theoretical guarantee. Others transform the subgraph matching problem into an edge matching problem and generate prediction results through the matching matrix obtained by Sinkhorn's algorithm (Roy et al., 2022), thereby providing interpretability for the model. The process of turning node matching into edge matching, however, loses necessary information about edges' relation, such as edges' common nodes, which hurts the expressibility of the model.

Graph Neural Networks (GNNs) are powerful techniques (Xu et al., 2019; Kipf & Welling, 2017) yielding breakthroughs in many key applications (Hamilton et al., 2017b). Graph neural networks mostly iteratively aggregate information that can be expressed as follows,

$$\begin{aligned} H^{(l+1)} &= \mathrm{AGG}_f(A, H^{(l)}) \\ H_{i:}^{(l+1)} &= \phi\left(H_{i:}^{(l)}, f\left(\left\{H_{j:}^{(l)} : j \in \mathcal{N}(i)\right\}\right)\right) \end{aligned} \tag{1}$$

where $f(\cdot)$ is the aggregation function such as mean or max; ϕ is the update function. $H^{(l)}$ is the representation matrix and its i-th row, denoted by $H_{i:}$, is the representation of node i. Over the last few years, there is considerable progress in proposing different ways of aggregating. For example, GraphSAGE (Hamilton et al., 2017b) aggregates node features with mean/max/LSTM pooled neighboring information. Graph Attention Network (GAT) (Velickovic et al., 2018) aggregates neighbor information using learnable attention weights. A close work for subgraph matching is Graph Isomorphism Network (GIN) (Xu et al., 2019), which converts aggregation as a learnable function based on the Weisfeiler-Lehman (WL) test to maximize the performance of GNNs. However, the WL test (Xu et al., 2019) cannot address the subgraph matching problem because it hashes the tree structure and ignores the partial order information for subgraph matching.

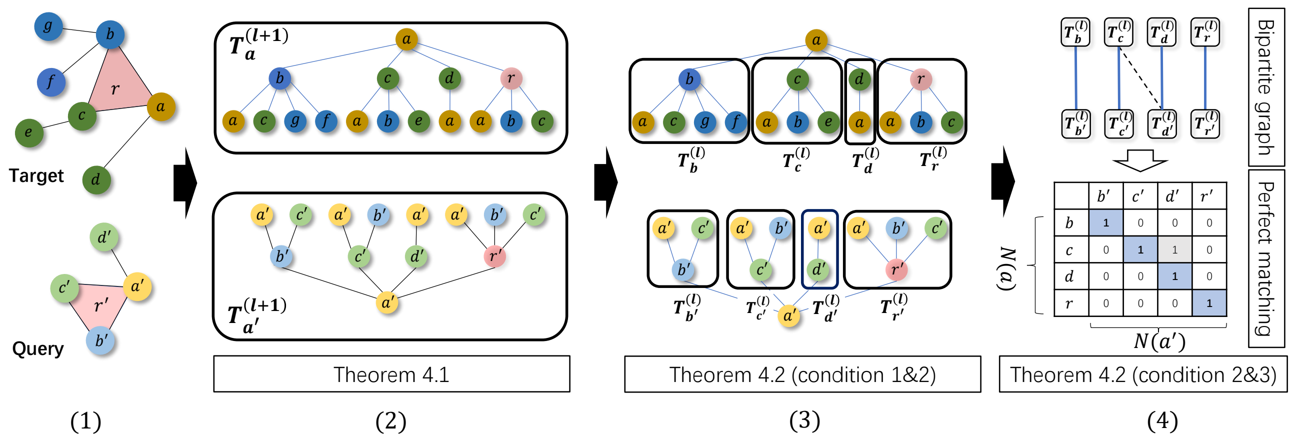

Target
Query
$T_a^{(l+1)}$
$T_{a'}^{(l+1)}$
$T_b^{(l)}$ $T_c^{(l)}$ $T_d^{(l)}$ $T_r^{(l)}$
$T_{b'}^{(l)}$ $T_{c'}^{(l)}$ $T_{d'}^{(l)}$ $T_{r'}^{(l)}$
Bipartite graph
Perfect matching
$N(a)$
$N(a')$
Theorem 4.1
Theorem 4.2 (condition 1&2)
Theorem 4.2 (condition 2&3)
(1)
(2)
(3)
(4)

Figure 1. Illustration of the proposed degeneracy procedure for subgraph matching: the problem of whether node a' in the query graph matches node a in the target one *degenerate* to check whether the constructed $(l+1)$-depth subtrees rooted at node a and a' are subtree isomorphic. This corresponds to Step (2) and is guaranteed by Theorem 4.1. The procedure is to check whether $T_a^{(l+1)}$ and $T_{a'}^{(l+1)}$ are subtree isomorphic. As shown in Step (3) and guaranteed by conditions 1&2 in Theorem 4.2, this is equivalent to checking whether subtree isomorphism holds for every l-depth subtrees rooted in $N(a')$ to a unique l-depth subtree in $N(a)$, where $N(\cdot)$ is the neighbor set of a given node. After that, based on conditions 2&3 in Theorem 4.2, the problem of subtree isomorphic is equivalent to checking whether there is perfect matching on the bipartite graph from every l-depth subtrees rooted in $N(a')$ and $N(a)$, respectively. This corresponds to Step (4), where the upper part represents the constructed bipartite graph, and the lower part is its adjacency matrix. By running the Hall's marriage algorithm, we can determine the perfect matching, where the selected edges are highlighted by the blue areas. The computation procedure for this indication matrix is implemented via the intrinsic tree-structured aggregation mechanism on GNNs, which is proved in Theorem 4.3 and guarantees the linear time cost as analyzed in Sec. 4.3.

3. Preliminary

We define some notations accordingly: Let $A_\mathcal{Q}$ and $A_\mathcal{T}$ be the adjacency matrix of the query graph $G_\mathcal{Q}$ and the target graph $G_\mathcal{T}$, respectively. $N(\cdot)$ denotes the neighbor set of a given node or a given set. $|\cdot|$ denotes the cardinality of a set. $T_v^{(l)}$ defines the subtree whose root is v and expands up to l-hop neighbors of v (or l-depth of the subtree). In the paper, the concepts of the l-hop neighbors and the l-depth subtrees are interchangeable.

Problem Definition: Suppose we are given a query graph, $G_\mathcal{Q}(V_\mathcal{Q}, E_\mathcal{Q})$, and a target graph, $G_\mathcal{T}(V_\mathcal{T}, E_\mathcal{T})$. Here, $(V_\mathcal{Q}, E_\mathcal{Q})$ and $(V_\mathcal{T}, E_\mathcal{T})$ are the pairs of vertices and edges related to the query graph and the target graph, respectively. Besides, the node attributes of the query graph and the target graph are denoted as $X_\mathcal{Q} \in \mathbb{R}^{|V_\mathcal{Q}|\times D}$ and $X_\mathcal{T} \in \mathbb{R}^{|V_\mathcal{T}|\times D}$, respectively, where D is the dimension of the node attributes. The problem of subgraph matching is to identify whether the query graph $G_\mathcal{Q}$ is subgraph isomorphic to the target graph $G_\mathcal{T}$, i.e. if there exists an injective $\xi : V_\mathcal{Q} \to V_\mathcal{T}$ such that $\forall u, v \in V_\mathcal{Q}, (u, v) \in E_\mathcal{Q} \Leftrightarrow (\xi(u), \xi(v)) \in E_\mathcal{T}$.

The injective can be represented by a matching matrix, $S \in \{0,1\}^{|V_\mathcal{T}|\times|V_\mathcal{Q}|}$, where $S_{ij} = 1$ if and only if node pair(i, j) is matched, i.e., $\xi(j) = i$. Therefore, subgraph isomorphism is equivalent to checking the existence of such a matrix S, i.e., whether there exists assignment matrix S such that:

$$SA_\mathcal{Q}S^T = A_\mathcal{T}, \quad \text{s.t.} \begin{cases} \sum_i S_{ij} = 1, \forall j \\ \sum_j S_{ij} \le 1, \forall i \\ S_{ij} \in \{0,1\}, \forall i, j. \end{cases} \tag{2}$$

We denote $G_\mathcal{Q} \subset G_\mathcal{T}$ if $G_\mathcal{Q}$ is a subgraph of $G_\mathcal{T}$. Here, we first define several key concepts:

WL Subtree: The Weisfeiler-Lehman (WL) test is an approximate solution to the graph isomorphism problem with linear computational complexity (Shervashidze et al., 2011). The WL test performs the aggregation on nodes' labels and their neighborhoods recursively, followed by hashing the aggregated results into unique new labels. As a result, this test produces an unordered tree for each node, called the WL subtree, which is a balanced tree with the height of the number of iterations. After repeating the algorithm k times, the obtained WL subtree for a node includes the structural information of the k-hop subgraph from that node. Research shows that the expressiveness of GNNs with the message-passing mechanism is upper-bounded by the WL test (Xu et al., 2019).

Subtree Generation: For any node v in a graph, one can obtain a subgraph $Sub_v^{(l)}$ by taking the l-hop neighbor of v. Given any tree generation method, e.g., the WL subtree, we always obtain its corresponding subtree whose root is v:

$$T_v^{(l)} = \Psi(Sub_v^{(l)}), \tag{3}$$

where Ψ is a subtree generation function. Unless stated otherwise, we employ the WL subtree to generate subtrees for a given node due to its uniqueness (Xu et al., 2018). Instead of explicitly constructing such trees, we can run GNNs in a graph, since building a k-order WL subtree is equivalent to aggregating k times in GNNs (Xu et al., 2018). Notice that traditional methods such as Breadth-First-Search (BFS) and Depth-First-Search (DFS) are not applicable at this work because they do not satisfy the uniqueness property. In particular, the tree generated for the same node by BFS or DFS will be different due to different search order.

Perfect Matching in Bipartite Graphs: A perfect matching (Gibbons, 1985) is a chosen edge set of a graph wherein every node of the graph is incident to exactly one edge. Hence, according to the edge set, each node in the graph corresponds to only one other node. The existence of a perfect matching on a bipartite graph can be resolved via Hall's Marriage Theorem (Hall, 1935).

Theorem 3.1. *(Hall's marriage theorem) Given a bipartite graph, $B(X, Y, E)$ that has two partitions: X and Y and $|Y| \leq |X|$, where E denotes the edges. The necessary and sufficient condition of the existence of the perfect matching in $B(X, Y, E)$ is : $\forall\ W \subseteq Y, |W| \leq |N(W)|$, where $N(W)$ is the neighborhood of W defined by $N(W) = \{b_j \in X : \exists a_i \in W, (a_i, b_j) \in E\}$.*

4. The Proposed Method

In the following, we present our proposed D^2Match with the theoretical derivation and its extensions.

4.1. Subgraph Matching Degeneracy

We approach the subgraph matching problem from a degeneracy perspective, framing this problem as a subtree matching problem with linear complexity. A fundamental question to the subgraph matching problem is on what conditions one subgraph is isomorphic to the other. Since the subgraph matching problem is NP-complete, the exact answer to this question becomes impractical. Instead, we can reduce the answer of finding both sufficient and necessary conditions to that of necessary only. What follows is to construct a criterion that any isomorphic pairs can meet.

Subtree isomorphism is a special task of subgraph isomorphism and can be attained in polynomial time. Here, we aim to reduce the subgraph isomorphism problem to a subtree isomorphism problem. Inspired by Xu et al. (2019), we construct a criterion by checking the subtrees rooted at the nodes: if $G_{\mathcal{Q}}$ is a subgraph of $G_{\mathcal{T}}$, then the tree rooted at any node q of $G_{\mathcal{Q}}$ should be a subtree of the tree rooted at the matched node $t = \xi(q)$ of $G_{\mathcal{T}}$. As stated in the following theorem, some additional properties of these trees are needed to make this criterion hold:

Theorem 4.1. *Given a target graph $G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}})$ and a query graph $G_{\mathcal{Q}}(V_{\mathcal{Q}}, E_{\mathcal{Q}})$, if $G_{\mathcal{Q}} \subset G_{\mathcal{T}}$, and the subtree generation function Ψ as defined in Eq. (3) meets the following condition:*

$$\forall \text{ graph pair } (G_{\mathcal{S}}, G), \text{if } G_{\mathcal{S}} \subset G, \text{then } \Psi(G_{\mathcal{S}}) \subset \Psi(G), \tag{4}$$

then the subgraph isomorphic mapping $\xi : V_{\mathcal{Q}} \rightarrow V_{\mathcal{T}}$ ensures the l-depth subtrees of the subgraph are isomorphic to the subtrees of the corresponding subgraph:

$$\forall l \geq 1, \mathrm{q} \in V_{\mathcal{Q}}, \mathrm{t} = \xi(\mathrm{q}) \in V_{\mathcal{T}} \Rightarrow T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}, \tag{5}$$

Please find the proof in Appendix A.1. This theorem provides a necessary condition for the potential isomorphic pairs. With this theorem, given a query graph $G_{\mathcal{Q}}$ and a target graph $G_{\mathcal{T}}$, the node q from $G_{\mathcal{Q}}$ is possible to match node t in $G_{\mathcal{T}}$ only if $T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}$. It then becomes an isomorphic test, which checks whether there exists an assignment such that each node q in $G_{\mathcal{Q}}$ possibly corresponds to one unique node t in $G_{\mathcal{T}}$. Forming a boolean indicator matrix $S^{(l)} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$:

$$S_{\mathrm{tq}}^{(l)} = \begin{cases} 1, \text{if } T_{\mathrm{q}} \subset T_{\mathrm{t}} \\ 0, \text{otherwise} \end{cases}, \tag{6}$$

we arrive at checking whether the indicator matrix $S^{(l)}$ contains a valid assignment matrix S satisfies Eq. (2).

Due to the favorite property of uniqueness, we employ the WL subtree as the generation function. What follows is how to determine the subtree isomorphism relationship between l-order trees. To be specific, based on WL subtree generation, we solve the subtree matching problem iteratively, intending to find a perfect matching on a bipartite graph in each iteration. The following theorem guarantees the conversion:

Theorem 4.2. *Given a node q in the query graph and a node t in the target graph, the following three conditions are equivalent:*

1) $T_{\mathrm{q}}^{(l+1)} \subset T_{\mathrm{t}}^{(l+1)}$.
2) There exists an injective function on the neighborhood of these nodes as $f : N(\mathrm{q}) \rightarrow N(\mathrm{t})$, s.t. $\forall \mathrm{q}_i \in N(\mathrm{q}), \mathrm{t}_i = f(\mathrm{q}_i), T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_i}^{(l)}$.
3) There exists a perfect matching on the bipartite graph $B^{(l)}(N(\mathrm{t}), N(\mathrm{q}), E)$, where $\forall \mathrm{t}_j \in N(\mathrm{t}), \mathrm{q}_i \in N(\mathrm{q}), (\mathrm{t}_j, \mathrm{q}_i) \in E$ if and only if $T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_j}^{(l)}$.

The proof is provided in Appendix A.2. The equivalence of the first two conditions implies that matching subtrees of a pair of nodes is equivalent to matching all subtrees from their child nodes. As a result, the indicator matrix can be updated recursively. That is, the indicator matrix at the $(l+1)$-th layer, i.e., $S^{(l+1)}$, should rely on $S^{(l)}$. Meanwhile, the equivalence of the last two conditions means that matching the subtrees from these child nodes is equivalent to solving the perfect matching on the corresponding bipartite graph whose nodes represent the subtrees of the child nodes. In summary, Theorem 4.2 tells us that subgraph matching is equivalent to delivering perfect matching on a bipartite graph. A visualization of this procedure is shown in Fig. 1.

Motivated by Hall's marriage Theorem 3.1, we develop an efficient algorithm to address the perfect matching procedure. A straightforward solution is to randomly select a subset W from the smaller partition of $B^{(l)}(N(\mathrm{t}), N(\mathrm{q}), E)$, i.e., $W \subseteq N(\mathrm{q})$, and count whether W's neighbors, i.e., $N(W) \subseteq N(\mathrm{t})$ in the other partition, have more elements than W. After repeating this process multiple times, we obtain a perfect matching when no instance violates the criterion. Note that we perform the procedure for all possible matched node pairs.

Is it possible to execute all pairs in parallel? Luckily, we can borrow GNNs to accomplish the perfect matching. Specifically, when computing a perfect matching between node $\mathrm{q} \in G_{\mathcal{Q}}$ and node $\mathrm{t} \in G_{\mathcal{T}}$, one needs to find W such that it satisfies $W \subseteq N(\mathrm{q})$ according to Theorem 3.1. In practice, we can obtain this by sampling the neighbors of node q, equating to sampling the edges, or the Drop Edge operation (Hamilton et al., 2017a). In this way, we obtain a sampled graph $\tilde{G}_{\mathcal{Q}}$ from the query graph $G_{\mathcal{Q}}$, along with its adjacency matrix $\tilde{A}_{\mathcal{Q}}$. Following Theorem 3.1, we conclude that $W = N'(\mathrm{q})$ with $N'(\mathrm{q}) \subset N(\mathrm{q})$ since node q's neighbors in $\tilde{G}_{\mathcal{Q}}$ are a subset of the original graph. At each iteration, we will perform the counting w.r.t. W and its neighbor set $N(W)$ for each node pair (t, q), and check whether $|N(W)| \geq |W|$ holds. To be efficient, we define a binary matrix, $\Phi^{(l)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}}) \in R^{|V_{\mathcal{T}}|\times|V_{\mathcal{Q}}|}$, where its element at (t, q) corresponds to the result of the node pair (t, q) between sampled query graph $\tilde{G}_{\mathcal{Q}}$ and target graph $G_{\mathcal{T}}$.

Based on Theorem 4.1, we need to update the indicator matrix $S^{(l)}$ recursively, where we compute $\Phi^{(l)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}})$ with multiple sampled $\tilde{G}_{\mathcal{Q}}$. We next show that computing Φ is equivalent to performing the GNN-based aggregation on the related graphs for any given $S^{(l)}$.

Theorem 4.3. *Given the sampled query graph and the target graph, we can construct their adjacency matrices , $\tilde{A}_{\mathcal{Q}}$ and $A_{\mathcal{T}}$, and the degree matrix of the sampled query graph $\tilde{D}_{\mathcal{Q}} = diag(\sum_s((\tilde{A}_{\mathcal{Q}})_{:s}))$. Here, we denote the indicator matrix at the l-th hop as $S^{(l)}$. To check the validity of $|N(W)| \geq |W|$ for each node pair, we can check whether each element of $\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}})$ is true or not, where $\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}}) := Z_{N(W)} \geq 1$, $Z_{N(W)} = AGG_{sum}(A_{\mathcal{T}}, Z_W^T)$ and $Z_W = AGG_{\max}(\tilde{D}_{\mathcal{Q}}^{-1} \cdot \tilde{A}_{\mathcal{Q}}, (S^{(l)})^T)$.*

The proof is provided in Appendix A.5. Recalling Theorem 3.1, we need to check $|N(W)| \geq |W|$ for each node pair (t, q), i.e., to check whether each element of Φ is true for each sampled $\tilde{A}_{\mathcal{Q}}^{(k)}$. The condition is valid only when Φ is true for each sample, i.e., $\tilde{G}_{\mathcal{Q}}^{(k)}$. Hence we can check the criterion by the following element-wise product:

$$S_{subtree}^{(l+1)} = \bigodot_{k=0}^{K} \Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}^{(k)}, A_{\mathcal{T}}), \tag{7}$$

where $\bigodot$ denotes the element-wise multiplication between matrices. In practice, $\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}^{(k)}, A_{\mathcal{T}})$ considers three cases:

$$\begin{cases} \mathrm{AGG}_{\mathrm{sum}}(A_{\mathcal{T}}, \mathrm{AGG}_{\max}(D_{\mathcal{Q}}^{-1}A_{\mathcal{Q}}, (S^{(l)}))^T) \geq 1 & k = 0 \\ \mathrm{AGG}_{\mathrm{sum}}(A_{\mathcal{T}}, \mathrm{AGG}_{\max}(\tilde{D}_{\mathcal{Q}}^{-1}\tilde{A}_{\mathcal{Q}}^{(k)}, (S^{(l)}))^T) \geq 1 & k \in [1, K) \\ \mathrm{AGG}_{\min}(A_{\mathcal{Q}}, \mathrm{AGG}_{\max}(A_{\mathcal{T}}, (S^{(l)})^T) \geq 1 & k = K \end{cases} \tag{8}$$

The above three cases allow us to balance the computation cost and accuracy. Initially, when $k = 0$, we deliver a full-size sampling for all nodes to avoid induction bias. When $k = K$, we perform the single-node sampling such that no node is omitted. The cases of $k \in [1, K-1]$ are computed via downsampling.

We want to highlight the difference between ours and other learning-based methods regarding GNNs. Here we employ a GNN model to accomplish the procedure of subtree matching, along with theoretical equivalence. Unlike performing the matching in our model, prior learning-based models use GNNs to capture the variance of data distribution for similarity inference since deep learning models learn distributional information to distinguish samples from different classes.

4.2. Boosting the Matching

It is noted that our D^2Match cannot guarantee that all isomorphism pairs are selected precisely. In order to boost the model performance, in the following, we design the corresponding mechanism to include more information, i.e., circle structures and node attributes to attain more precise isomorphism pairs.

4.2.1. Dealing with Circles

Prior methods often ignore circle structures though they are common in graphs and critical to be handled. In particular, learning-based methods rely on the expressibility of GNNs, which have difficulty identifying cyclic structures due to their WL-tree like aggregation mechanism (Sato, 2020). The underlying idea of our D^2Match is to select certain set of circles and construct the circle structures as *supernodes*. This allows us to formulate the circle matching as a standard subtree matching problem. Before detailing our strategy, we first present two desired properties of the selected set of circles in a graph.

Atomic: Let $c = (v_1, ..., v_{l(c)}, v_1)$ define a circle and $v(c)$ be the set of nodes of circle c, a circle is an *atomic* circle if it does not contain a smaller circle. That is, there is no circle c' such that $v(c') \subset v(c)$. A circle set $\mathcal{C}$ is *atomic* if every circle in the set is *atomic*.

Consistency: Each query circle must correspond to one circle in the target graph if the query graph is a subgraph of the target graph , i.e., $\forall c_\mathcal{Q} = (v_1, ..., v_l, v_1) \in \mathcal{C}_\mathcal{Q}, \exists c_\mathcal{T} = (\xi(v_1), ..., \xi(v_l), \xi(v_1)) \in \mathcal{C}_\mathcal{T}$ where ξ is the subgraph isomorphic mapping. $\mathcal{C}_\mathcal{Q}$ and $\mathcal{C}_\mathcal{T}$ are the selected circle sets of the query graph and the target graph, respectively.

The atomic property aims to ensure the compactness of circles, and the consistency attempts to ensure that the relation between a query and a target set of circles is injective. These two properties ensure a well-qualified set for matching. In practice, we can take advantage of *chordless cycles* (West, 2000), to serve our goal of matching circles. We now state our theorem below to show that these cycles satisfy the above consistency and atomic property.

Theorem 4.4. *Every chordless cycle is atomic. Every chordless cycle $c_\mathcal{Q}$ in an induced subgraph of the original query graph $G_\mathcal{Q}$ must correspond to a chordless cycle $c_\mathcal{T}$ in the origin graph $G_\mathcal{T}$.*

The proof is provided in Appendix, A.6. This theorem suggests that chordless cycles satisfy the above two properties, making them suitable for representing circles in a graph. To match circles, we introduce an augmented graph by inserting supernodes that embody these circles. Given a length L, we can acquire corresponding chordless cycles whose length is no longer than L for the query and target graphs as : $\mathcal{C}_\mathcal{T} = \{c|l(c) \leq L, c \in CC_\mathcal{T}\}, \mathcal{C}_\mathcal{Q} = \{c|l(c) \leq L, c \in CC_\mathcal{Q}\}$ where $CC_\mathcal{T}$ and $CC_\mathcal{Q}$ are the chordless cycle set. By setting v_c as the supernode of any chordless circle c, we connect nodes from the circle c to this supernode, resulting in an augmented graph. Note that supernodes can only match other supernodes to keep the matching of non-circles untainted. To this end, we transform the circle matching as the subtree matching problem such that we can employ the proposed method on the augmented graph directly. Unless otherwise stated, we keep all notations the same in the augmented graph to avoid abusing the notations.

4.2.2. Dealing with Nodes' Attributes

Apart from circle structures, subgraph matching also involves node attributes. Within the context of subgraph matching, learning with node attributes alone may be misleading because these cannot catch structural isomorphism. As a result, we employ the obtained subtree indicator matrix to supervise the learning process, aiming to filter out pairs that do not pass the test in the subtree matching. We are thus motivated to enhance the node attributes by concatenating the subtree matching indicator, resulting in the node representation for the query and target graphs as follows:

$$\begin{cases} H_\mathcal{T}^{(l+1)} = GNN_\mathcal{T}^{(l)}(A_\mathcal{T}, \text{concat}(H_\mathcal{T}^{(l)}, MLP(S^{(l)}))) \\ H_\mathcal{Q}^{(l+1)} = GNN_\mathcal{Q}^{(l)}(A_\mathcal{Q}, \text{concat}(H_\mathcal{Q}^{(l)}, MLP(S^{(l)})^T))) \end{cases} \quad (9)$$

Here, we employ an MLP model to reduce the effect of the difference between the node attributes and the indicator matrix, where the latter behaves as a one-hot encoding feature. We concatenate each pair of representations and then pass it to the MLP to obtain their similarity. For the node pair (i, j), we computer their similarity as

$$[S_{gnn}^{(l+1)}]_{ij} = MLP(\text{concat}([H_\mathcal{T}^{(l)}]_i, [H_\mathcal{Q}^{(l)}]_j)). \quad (10)$$

Now, we obtain a generalized indicator matrix that includes both the structure information and node attribute information

$$S^{(l+1)} = S_{gnn}^{(l+1)} \odot S_{subtree}^{(l+1)} \quad (11)$$

4.3. Implementation Details and Complexity Analysis

We summarize the whole procedure in Algorithm 1 outlined in Appendix A.1. We now analyze the time cost of our D^2Match. It is noted that our D^2Match consists of two major parts: the subtree module and the GNN module. Given L layers and K times of sampling, the complexity of the subtree module and the GNN module is $O(L * K * |V_\mathcal{T}| * |E_\mathcal{Q}| + L * |V_\mathcal{Q}| * |E_\mathcal{T}|)$ (according to the computation of lines 4-9 in Algo. 1) and $O(L * |E_\mathcal{T}| + L * |E_\mathcal{Q}| + |V_\mathcal{T}| * |V_\mathcal{Q}|)$ (according to the computation of lines 10-12 in Algo. 1), respectively. Since the query graph is often very small, we can treat $|V_\mathcal{Q}|$ and $|E_\mathcal{Q}|$ as constants. Therefore, the overall complexity is reduced to $O(|V_\mathcal{T}| + |E_\mathcal{T}|)$, attaining the linear time complexity. Please refer to Appendix A.3 for more details about the implementation. We also provide an empirical runtime comparison in Appendix A.7.

5. Experiments

Here, we conduct extensive experiments to address the following questions: (1) *How does our D^2Match perform comparing to SOTA GNN-based methods?* (2) *Why does our D^2Match work better than other GNN-based methods?* (3) *How does our D^2Match perform comparing to heuristic CO-based methods?* (4) *What is the effect of our D^2Match on the hyperparameters?* (5) *What is the convergence behavior of our D^2Match?* Sec. 5.2-Sec. 5.6 answer the above questions accordingly.

5.1. Experimental Settings

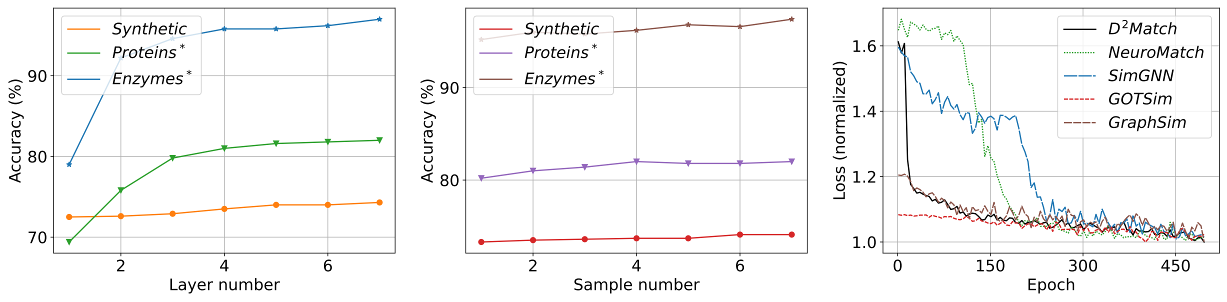
Datasets and Experimental Setup. We implement our experiments on both synthetic and real-world datasets, which are collected from a large variety of applications. We aim to

Table 1. Overall performance comparison in terms of accuracy.

	Synthetic	Proteins	Mutag	Enzymes	Aids	IMDB-Binary	Cox2	FirstMMDB
SimGNN (Bai et al., 2019)	70.5$_{\pm 2.72}$	96.2$_{\pm 0.97}$	98.7$_{\pm 0.60}$	98.6$_{\pm 1.08}$	96.5$_{\pm 0.68}$	85.0$_{\pm 19.58}$	99.9$_{\pm 0.22}$	82.40$_{\pm 0.17}$
NeuroMatch (Rex et al., 2020)	65.7$_{\pm 8.98}$	94.5$_{\pm 1.73}$	99.2$_{\pm 0.22}$	97.9$_{\pm 1.08}$	97.4$_{\pm 0.96}$	86.5$_{\pm 6.51}$	100.0$_{\pm 0.00}$	80.80$_{\pm 0.39}$
IsoNet (Roy et al., 2022)	50.0$_{\pm 0.00}$	60.0$_{\pm 10.02}$	94.1$_{\pm 2.54}$	91.0$_{\pm 7.78}$	61.5$_{\pm 8.51}$	83.1$_{\pm 3.69}$	95.8$_{\pm 3.89}$	/
GMN-embed (Li et al., 2019)	56.6$_{\pm 8.61}$	93.8$_{\pm 2.41}$	90.8$_{\pm 6.16}$	89.4$_{\pm 16.44}$	78.3$_{\pm 6.92}$	69.3$_{\pm 15.18}$	69.7$_{\pm 18.20}$	69.1$_{\pm 30.29}$
GraphSim (Bai et al., 2020)	50.0$_{\pm 0.00}$	82.5$_{\pm 0.31}$	89.5$_{\pm 2.59}$	88.2$_{\pm 1.79}$	75.6$_{\pm 6.53}$	88.9$_{\pm 2.81}$	95.5$_{\pm 0.94}$	86.6$_{\pm 9.71}$
GOT-Sim (Doan et al., 2021)	53.0$_{\pm 2.74}$	57.2$_{\pm 8.52}$	86.8$_{\pm 6.92}$	68.7$_{\pm 14.15}$	70.6$_{\pm 3.08}$	81.3$_{\pm 14.60}$	94.8$_{\pm 1.04}$	/
D^2Match	**74.3**$_{\pm 0.22}$	**100.0**$_{\pm 0.00}$	**100.0**$_{\pm 0.00}$	**99.9**$_{\pm 0.22}$	**99.5**$_{\pm 0.27}$	**93.3**$_{\pm 1.03}$	**100.00**$_{\pm 0.00}$	**100.00**$_{\pm 0.00}$

Table 2. Results of experimenting the uniformly distributed data in terms of accuracy.

	Synthetic$^+$	Synthetic	Proteins*	Mutag*	Enzymes*	Aids*	IMDB-Binary*	Cox2*	FirstMMDB*
SimGNN	84.1$_{\pm 3.40}$	70.5$_{\pm 2.72}$	64.6$_{\pm 3.36}$	80.3$_{\pm 6.18}$	76.0$_{\pm 2.12}$	73.2$_{\pm 6.06}$	72.0$_{\pm 2.45}$	88.2$_{\pm 2.05}$	53.2$_{\pm 6.61}$
NeuroMatch	74.5$_{\pm 2.57}$	65.7$_{\pm 8.98}$	52.8$_{\pm 4.76}$	90.4$_{\pm 2.88}$	86.6$_{\pm 3.64}$	75.6$_{\pm 17.78}$	60.4$_{\pm 10.11}$	91.0$_{\pm 5.70}$	50.0$_{\pm 0.00}$
D^2Match	**86.6**$_{\pm 1.44}$	**74.3**$_{\pm 0.22}$	**83.4**$_{\pm 2.97}$	**99.2**$_{\pm 0.84}$	**96.0**$_{\pm 2.16}$	**95.0**$_{\pm 1.41}$	**90.2**$_{\pm 1.79}$	**99.8**$_{\pm 0.45}$	**86.4**$_{\pm 7.44}$



(a) # layers vs. accuracy (b) # sampling vs. accuracy (c) The convergence comparison

Figure 2. We conduct sensitivity analysis on our D^2Match by varying the number of layers and sampling. In Fig. 2(c), we present the convergence curve on our D^2Match and four strong baselines.

obtain pairs of query and target graphs, along with labels indicating whether a query is isomorphic to the target. We first generate synthetic data by utilizing ER-random graphs and WS-random graphs (Rex et al., 2020). We keep edge densities the same in both positive and negative samples to ensure consistency in the distribution. This balance avoids potential biases during learning. For the real-world data, we follow the setting in (Rex et al., 2020), including Cox2, Enzymes, Proteins, IMDB-Binary, MUTAG, Aids, and FirstMMDB. Please refer to Appendix A.9 for additional experiments, such as Open Graph Benchmark datasets(Hu et al., 2020).

We employ these raw graphs as target graphs and generate the positive query graphs using random breadth-first search sampling from the target graphs. The negative query graphs are randomly generated. Similar to the synthetic data, we require the edge density in both positive and negative samples to be as close as possible. We split each dataset into training and testing at a ratio of $4:1$ and report the average classification accuracy under the five-fold cross-validation.

Baselines. For a fair comparison, we select the following SOTA GNN-based competitors: SimGNN (Bai et al., 2019), NeuralMatch (Rex et al., 2020), IsoNet (Roy et al., 2022), GMN-embed (Li et al., 2019), GraphSim (Bai et al., 2020), and GOT-Sim (Doan et al., 2021). These all incorporate graph neural networks into subgraph matching. For combinatorial optimization metheds, we choose GQL (He & Singh, 2008), QSI (Shang et al., 2008), CECI (Bhattarai et al., 2019) and set-intersection method LFTJ (Sun & Luo, 2020). These are traditional methods that can find exact solutions.

5.2. Main Results

Table 1 reports the overall performance of all compared models, where each model achieves the best results in all possible settings up to 500 epochs. The accuracy of GOT-Sim and IsoNet on FirstMMDB is omitted due to exceeding time and memory. We observe that

- For the synthetic dataset, the overall performance is much lower than that in the real-world datasets. A reason is that the synthetic dataset is more complicated, e.g., with a higher edge density, than real-world datasets, which makes the matching more challenging. By examining more details, IsoNet, GMN-embed, GraphSim, and GOT-Sim attempt to employ a node-level assignment matrix to capture matching between graphs, which underestimates the importance of global structure. They can only yield around 50% accuracy. SimGNN and NeuroMatch try to learn the global representation and attain the accuracy of 70.5% and 65.7%, respectively.

- For the real-world datasets, D^2Match attains superior performance and beats all baselines. Among seven real-world datasets, D^2Match attains 100% accuracy in four datasets, i.e., Protein, Mutag, Cox2, and FirstMMDB, while at least 99.5% in Enzymes and Aids, and 93.3% in IMDB-Binary.
- Overall, D^2Match has explicitly modeled subtrees and consistently attained the best performance among all compared methods. The promising results confirm our theoretical analysis.

Table 3. Time comparison with CO-based metheds(seconds).

	Proteins	Mutag	Enzymes	Aids	Cox2
GQL	3.82	2.05	2.98	6.94	3.34
QSI	10.37	**0.16**	4.63	19.67	3.22
CECI	19.74	0.27	13.74	19.15	6.19
LFTJ	9.93	0.21	4.61	20.46	5.69
D^2Match	**1.12**	1.04	**1.18**	**1.88**	**1.68**

Table 4. Efficiency comparison with CO-based metheds.

	Proteins	Mutag	Enzymes	Aids	Cox2
GQL	96.2	100.0	**100.0**	**100.0**	100.0
QSI	96.8	100.0	99.8	94.8	100.0
CECI	91.2	100.0	99.4	91.0	100.0
LFTJ	87.0	100.0	89.8	95.8	100.0
D^2Match	**100.0**	**100.0**	99.9	99.5	**100.0**

5.3. Benefit of Our Core Design: The Subtree Matching

The following experiments verify how GNNs deployed in our D^2Match function differently in existing GNN-based subgraph matching methods. That is, our D^2Match utilizes GNNs to explicitly model subtrees while existing methods optimize the graph representations via memorizing the data distribution divergence. We additionally construct new datasets and denote them with *. The new datasets exclude the data distribution effect by the following steps: following the same way to generate the positive samples as in Sec. 5.1, continuing to perform edge dropping and insertion on the clipped graphs together to obtain the final negative samples. This strategy aims to make sure the generated samples follow almost the same distribution in terms of edges. For the synthetic dataset, given that the positive and negative data in the original dataset are generated randomly and following the same distribution, we construct another synthetic dataset without following the same density distribution for better comparison. Since SimGNN and NeuoMatch are the best-performing GNN-based methods in Table 1, we compare our D^2Match with them.

Results in Table 2 show that D^2Match outperforms SimGNN and NeuoMatch by a much larger margin, achieving $2.5\% - 33.2\%$ improvement. This phenomenon aligns with our hypothesis that the gain of other learning-based methods is distribution-dependent, which results in a significant performance drop on evenly-distributed data. Moreover, the overall performance on Synthetic$^+$ is much better than that on Synthetic. This implies that data following non-even distribution will make the matching much easier. This is in line with the results in Table 1, i.e., the GNN-based methods tend to capture the distribution divergence rather than performing matching.

5.4. Comparison with exact combinatorial solutions

Before detailing the comparison between D^2Match and CO-based methods, we would like to emphasize that our method and CO-based heuristics are driven by different purposes and are therefore suitable for different scenarios. Heuristic algorithms can find exact solutions, but the worst-case time complexity is exponential, and they are generally suitable for scenarios with high matching accuracy requirements, such as graph databases. Our method is suitable for efficient machine learning tasks with time constraints. To verify the advantages of our method, we compared it from three perspectives:

Solution time: Since heuristic algorithms can find exact solutions, the metric of accuracy is no longer distinguishable. Therefore, we mainly compared our algorithm and heuristic algorithms in terms of the average running time on the original dataset. We report the average running time of our method and heuristic algorithms on the test set, as shown in Table 3. The results demonstrate that our method not only has a consistently stable running time, but it is also significantly faster than heuristic algorithms on most datasets, often by an order of magnitude.

Solution efficiency: To compare the efficiency of different solutions, we compared the success rates of heuristic solvers to the accuracy of our method, shown in Table 4. The success rate is defined as the proportion of samples that an algorithm completes within 10 seconds per sample on the original dataset. Considering the constraint of execution time, our method's accuracy and heuristic methods' success rate are comparable to those based on heuristic algorithms for most real datasets.

Scalability: To validate the scalability of our model, we conducted experiments using larger datasets. Specifically, we used the ”firstmm” dataset consisting of 50,000 nodes, as well as the ”DD” dataset consisting of 300,000 nodes and one million edges. We compared our method with these datasets in terms of runtime and success rate. As shown in Table 5, our method demonstrates better efficiency than heuristic algorithms on large-scale graphs. On the DD dataset, our method achieves an accuracy rate of nearly 100%. On the firstmm dataset, our method has a relatively low accuracy rate, but it is comparable to some heuristic algorithms. These results suggest that our method is scalable.

Table 5. Results of experimenting the scalability.

	Time(seconds)		Success rate(%)	
	DD	Firstmm	DD	Firstmm
GQL	58.49	149.15	**100.0**	60.0
QSI	80.18	127.21	94.0	60.4
CECI	128.93	163.73	48.4	69.6
LFTJ	80.36	89.85	92.8	**77.2**
D^2Match	**46.61**	**32.62**	99.2	60.0

Furthermore, we included the success rates of our heuristic methods on synthetic datasets with varying numbers of nodes. This demonstrates the scalability of our method on synthetic data, as shown in Table 6. In general, the success rate of heuristic methods decreases rapidly as the scale increases. However, our method has consistently maintained an accuracy of around 70%.

Table 6. Results of synthetic datasets.

Nodes	20	40	60	80	100	200
GQL	**75.0**	**81.8**	**78.0**	63.2	56.8	32.0
QSI	70.4	61.6	19.4	12.2	11.2	2.6
CECI	66.0	51.2	13.8	12.4	15.0	16.8
LFTJ	60.2	71.4	25.6	15.8	12.0	5.2
D^2Match	70.0	70.2	74.0	**63.4**	**71.0**	**67.0**

5.5. Sensitivity Analysis

Effect of L, the depth of a subtree. We test the effect of the depth of a subtree, i.e., the number of the hidden layers, and change it from 1 to 7. Results in Fig. 2(a) show that D^2Match reaches its best performance when the number of layers is 6. D^2Match only needs a few layers to achieve decent performance, suggesting it can scale up to large size graphs.

Effect of K, the times of samples. Intuitively, sampling more data to train the model will yield better performance. We vary K from 1 to 7 and show the results in Fig. 2(b). Surprisingly, the results show that by sampling five times, we can obtain the best performance on all datasets. This demonstrates that D^2Match can attain decent performance at a low computation cost.

We also ablate D^2Match with circles and node attributes at different settings, and all experiments show results consistent with our theoretical analysis. Please refer to Appendix A.5 for more details.

5.6. Convergence Analysis

Figure 2(c) provides the training loss of D^2Match and four baselines on Synthetic, where we only select baselines with the same loss functions as ours, such as MSE or CE, for a fair comparison. The results show that (1) D^2Match converges the fastest due to its power of explicitly modeling the subtrees. (2) NeuroMatch and SimGNN perform matching through learning graph-level representations, which need more epochs to converge for capturing the local structure. (3) GOTSim and GraphSim attain the lowest loss in the beginning but show the weakest convergence ability compared to others because they can only capture the node-level representations and fail to learn meaningful subgraph matching. Consequently, they yield the worst performance as reported in Table 1.

6. Conclusion and Future work

In this paper, we propose D^2Match for subgraph matching, which degenerates the subgraph matching problem into perfect matching in a bipartite graph and proves that the matching procedure can be implemented via the built-in tree-structure aggregation on GNNs, which yields linear time complexity. We also incorporate circle structures and node attributes to boost the matching performance. Finally, we conduct extensive experiments to show that D^2Match achieves significant improvement over competitive baselines and indeed exploits subtrees for the matching, which is different from existing GNN-based methods for memorizing the data distribution divergence.

D^2Match can be further explored in several promising directions. First, we can investigate more degeneracy mechanisms to tackle more complicated graphs. Second, we can exploit other information, e.g., positional encoding, to boost the model performance. Third, we can extend our D^2Match to more real-world applications, e.g., document matching, to investigate its capacity.

7. Acknowledgement

The work was fully supported by the IDEA Information and Super Computing Centre (ISCC) and was partially supported by the National Nature Science Foundation of China (No. 62201576), the National Key Research and Development Program of China (No. 2020YFB1708200), the ”Graph Neural Network Project” of Ping An Technology (Shenzhen) Co., Ltd. and AMiner.Shenzhen SciBrain fund.

References

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., and Wang, W. Simgnn: A neural network approach to fast graph similarity computation. WSDM '19, pp. 384–392, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405.

Bai, Y., Ding, H., Gu, K., Sun, Y., and Wang, W. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3219–3226, 04 2020.

Bhattarai, B., Liu, H., and Huang, H. H. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pp. 1447–1462, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450356435. doi: 10.1145/3299869.3300086. URL https://doi.org/10.1145/3299869.3300086.

Caelli, T. and Kosinov, S. An eigenspace projection clustering method for inexact graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26 (4):515–519, 2004.

Chen, L., Gan, Z., Cheng, Y., Li, L., Carin, L., and Liu, J. Graph optimal transport for cross-domain alignment. 2020.

Cordella, L., Foggia, P., Sansone, C., and Vento, M. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

Doan, K. D., Manchanda, S., Mahapatra, S., and Reddy, C. K. Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, pp. 665–674, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020. URL https://arxiv.org/abs/2003.00982.

Gibbons, A. *Algorithmic graph theory*. Cambridge university press, 1985.

Hall, P. On representatives of subsets. *Journal of The London Mathematical Society-second Series*, pp. 26–30, 1935.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017a. URL http://arxiv.org/abs/1706.02216.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017b.

Han, W.-S., Lee, J., and Lee, J.-H. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD '13*, 2013.

He, H. and Singh, A. K. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 405–418, 2008.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., and Tao, S. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 901–912, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306614.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kosinov, S. and Caelli, T. Inexact multisubgraph matching using graph eigenspace and clustering models. pp. 133–142, 08 2002. ISBN 978-3-540-44011-6.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3835–3845. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/li19d.html.

McCreesh, C., Prosser, P., Solnon, C., and Trimble, J. When subgraph isomorphism is really hard, and why this matters for graph databases. *J. Artif. Int. Res.*, 61(1):723–759, jan 2018. ISSN 1076-9757.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. Network motifs: Simple building blocks of complex networks. *Science (New York, N.Y.)*, 298:824–7, 11 2002.

Mongiovì, M., Natale, R. D., Giugno, R., Pulvirenti, A., Ferro, A., and Sharan, R. Sigma: a set-cover-based inexact graph matching algorithm. *Journal of bioinformatics and computational biology*, 8 2:199–218, 2010.

Peng, H., Li, J., Gong, Q., Ning, Y., Wang, S., and He, L. Motif-matching based subgraph-level attentional convolutional network for graph classification. In *AAAI*, 2020.

Priestley, H. A. and Ward, M. P. A multipurpose backtracking algorithm. *J. Symb. Comput.*, 18(1):1–40, 1994. URL http://dblp.uni-trier.de/db/journals/jsc/jsc18.html#PriestleyW94.

Rex, Ying, Lou, Z., You, J., Wen, C., Canedo, A., and Leskovec, J. Neural subgraph matching, 2020.

Roy, I., Velugoti, V. S. B. R., Chakrabarti, S., and De, A. Interpretable neural subgraph matching for graph retrieval. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8115–8123, Jun. 2022.

Sato, R. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

Shang, H., Zhang, Y., Lin, X., and Yu, J. X. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. 1(1):364–375, aug 2008. ISSN 2150-8097.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Sun, S. and Luo, Q. In-memory subgraph matching: An in-depth study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pp. 1083–1098, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3380581. URL https://doi.org/10.1145/3318464.3380581.

Ullmann, J. R. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, jan 1976. ISSN 0004-5411.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Vesselinova, N., Steinert, R., Perez-Ramirez, D. F., and Boman, M. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

West, D. B. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000. ISBN 0130144002.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICML*, 2019.

Yan, X., Yu, P. S., and Han, J. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pp. 766–777, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930604.

Zhou, F. and Torre, F. Canonical time warping for alignment of human behavior. *Advances in neural information processing systems*, 22, 2009.

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *CoRR*, abs/1707.04638, 2017. URL http://arxiv.org/abs/1707.04638.

A. Appendix

A.1. The Pseudo-Code of D^2Match

The pseudo-code of D^2Match is outlined as follows:

Algorithm 1 The D^2Match algorithm

```
Require: Query graph $G_{\mathcal{Q}}(V_{\mathcal{Q}}, E_{\mathcal{Q}})$ with node attributes $X_{\mathcal{Q}}$, target graph $G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}})$ with node attributes $X_{\mathcal{T}}$, iteration number: $L$, sample number: $K$.
Ensure: Is $G_{\mathcal{Q}}$ isomorphic to $G_{\mathcal{T}}$
 1: $G_{\mathcal{Q}}(V_{\mathcal{Q}}, E_{\mathcal{Q}}) \leftarrow ChordlessCycleAugment(G_{\mathcal{Q}})$;
    $G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}}) \leftarrow ChordlessCycleAugment(G_{\mathcal{T}})$;
 2: $H_{\mathcal{Q}}^{(0)} = X_{\mathcal{Q}} \in \mathbb{R}^{|V_{\mathcal{Q}}| \times D}$; $H_{\mathcal{T}}^{(0)} = X_{\mathcal{T}} \in \mathbb{R}^{|V_{\mathcal{T}}| \times D}$;
 3: $S^{(0)} = InitialAssignMatrix(X_{\mathcal{T}}, X_{\mathcal{Q}}) \in \mathbb{R}^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$
 4: for $l = 0, 1..., L-1$ do
 5:   for $k = 0, 1, ..., K$ do
 6:     $\tilde{A}_{\mathcal{Q}}^{(k)} = DropEdge(A_{\mathcal{Q}}) \in \mathbb{R}^{|V_{\mathcal{Q}}| \times |V_{\mathcal{Q}}|}$;
 7:     Calculate $\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}^{(k)}, A_{\mathcal{T}})$ according to Eq. (8)
 8:   end for
 9:   $S_{subtree}^{(l+1)} = \bigodot_{k=0}^{K} \Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}^{(k)}, A_{\mathcal{T}}) \in \mathbb{R}^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$;
10:   $H_{\mathcal{T}}^{(l+1)} = GNN_{\mathcal{T}}^{(l)}(A_{\mathcal{T}}, concat[H_{\mathcal{T}}^{(l)}, MLP(S^{(l)})]) \in \mathbb{R}^{|V_{\mathcal{T}}| \times d}$;
11:   $H_{\mathcal{Q}}^{(l+1)} = GNN_{\mathcal{Q}}^{(l)}(A_{\mathcal{Q}}, concat[H_{\mathcal{Q}}^{(l)}, MLP((S^{(l)})^T)]) \in \mathbb{R}^{|V_{\mathcal{Q}}| \times d}$;
12:   Compute $S_{gnn}^{(l+1)} \in \mathbb{R}^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$ according to Eq.10
13:   $S^{(l+1)} = S_{gnn}^{(l+1)} \odot S_{subtree}^{(l+1)}$;
14: end for
15: $result = CheckAssign(S^{(L)})$
```

A.2. Main Results

Theorem A.1. *Given a target graph $G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}})$ and a query graph $G_{\mathcal{Q}}(V_{\mathcal{Q}}, E_{\mathcal{Q}})$, if $G_{\mathcal{Q}} \subset G_{\mathcal{T}}$, and the subtree generation function Ψ as defined in Eq. (3) meets the following condition:*

$$\forall \textit{ graph pair } (G_S, G), \textit{if } G_S \subset G, \textit{then } \Psi(G_S) \subset \Psi(G), \tag{12}$$

then the subgraph isomorphic mapping $\xi : V_{\mathcal{Q}} \to V_{\mathcal{T}}$ ensures the l-hop subtrees of the subgraph is isomorphic to the subtrees of the corresponding subgraph:

$$\forall l \geq 1, \mathrm{q} \in V_{\mathcal{Q}}, \mathrm{t} = \xi(\mathrm{q}) \in V_{\mathcal{T}} \Rightarrow T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}, \tag{13}$$

Proof. According to the definition of subgraph matching (McCreesh et al., 2018), when $G_{\mathcal{Q}}$ is a subgraph of $G_{\mathcal{T}}$, there must exists an injective function $\xi : V_{\mathcal{Q}} \to V_{\mathcal{T}}$, such that $\forall \mathrm{q}_i, \mathrm{q}_j \in V_{\mathcal{Q}}, (\mathrm{q}_i, \mathrm{q}_j) \in E_{\mathcal{Q}} \Rightarrow (\xi(\mathrm{q}_i), \xi(\mathrm{q}_j)) \in E_{\mathcal{T}}$. For any subgraph in the query graph, e.g., $S(V_S, E_S) \in G_{\mathcal{Q}}$, we always have a subgraph in the original graph $G_{\mathcal{T}}$, denoted as $G_S(V_G, E_G)$, that corresponds to the set of the query node as $V_G = \xi(V_S)$. This tells us that $S \subset G_S$. According to this, consider any given node from $V_{\mathcal{Q}}$: $\mathrm{q} \in V_{\mathcal{Q}}$, $S_{\mathrm{q}}^{(l)}$ is a subgraph of $G_{\mathcal{Q}}$ and its image $G_{S_{\mathrm{q}}^{(l)}}$ in $G_{\mathcal{T}}$, i.e. $S_{\mathrm{q}}^{(l)} \subset G_{S_{\mathrm{q}}^{(l)}}$. By definition, the node in $S_{\mathrm{q}}^{(l)}$ or $G_{S_{\mathrm{q}}^{(l)}}$ is at most l-hop from node q or $\mathrm{t} = \xi(\mathrm{q})$, we know that $G_{S_{\mathrm{q}}^{(l)}}$ must be a subgraph of $S_{\mathrm{t}}^{(l)}$, i.e.,$G_{S_{\mathrm{q}}^{(l)}} \subset S_{\mathrm{t}}^{(l)}$. Put all together, we have $S_{\mathrm{q}}^{(l)} \subset G_{S_{\mathrm{q}}^{(l)}} \subset S_{\mathrm{t}}^{(l)}$. Based on the listed constrain, we then have $T_{\mathrm{q}}^{(l)} \subset T_{\mathrm{t}}^{(l)}$.

□

Theorem A.2. *Given a node* q *in the query graph and a node* t *in the target graph, the following three conditions are equivalent:*

1) $T_{\mathrm{q}}^{(l+1)} \subset T_{\mathrm{t}}^{(l+1)}$.

2) There exists an injective function on the neighborhood of these nodes as $f : N(\mathrm{q}) \to N(\mathrm{t})$, s.t. $\forall \mathrm{q}_i \in N(\mathrm{q}), \mathrm{t}_i = f(\mathrm{q}_i), T_{\mathrm{q}_i}^{(l)} \subset T_{\mathrm{t}_i}^{(l)}$.

3) There exists a perfect matching on the bipartite graph $B^{(l)}(N(\mathrm{t}), N(\mathrm{q}), E)$, where $\forall \mathrm{t}_j \in N(\mathrm{t}), \mathrm{q}_i \in N(\mathrm{q}), (\mathrm{t}_j, \mathrm{q}_i) \in E$ if and only if $T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_j}$.

We prove this theorem by introducing the following two theorem. Theorem A.3 shows that condition 1) is equivalent to condition 2), i.e. the WL subtree isomorphism test can be accomplished in a recursive manner then prove Theorem. A.4 that the condition 2) equals to condition 3) which means every iteration in the recursive process equals to examine the existence of a perfect matching, respectively.

Theorem A.3. *Given a node* q *in the query graph and a node* t *in the target graph, the following two conditions are equal:*
1) $T^{(l+1)}_{\mathrm{q}} \subset T^{(l+1)}_{\mathrm{t}}$, where l is an integer and $l \geq 1$.
2) There exists an injective function on the neighboring set of these nodes as $f : N(\mathrm{q}) \rightarrow N(\mathrm{t}), s.t. \forall \mathrm{q}_i \in N(\mathrm{q}), \mathrm{t}_i = f(\mathrm{q}_i), T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_i}$.

Proof. We assume f_{q} is a subtree isomorphism injective function in the condition 1), that $\forall$ node $u, v \in T^{(l+1)}_{\mathrm{q}}$, (u, v) is an edge of $T^{(l+1)}_{\mathrm{q}} \Rightarrow ((f_{\mathrm{q}}(u), f_{\mathrm{q}}(v))$ is an edge of $T^{(l+1)}_{\mathrm{t}}$. Similarly We also assume f_{q_i} is subtree isomorphism injective in the condition 2).
On the one hand, if condition 1) is true then f_{q} exists. Using the property of WL tree, we have $\forall \mathrm{q}_i \in N(\mathrm{q}), T^{(l)}_{\mathrm{q}_i} \subset T^{(l+1)}_{\mathrm{q}}$, which means the l-order WL tree of any node q_i in q's neighbourhood belongs to the $l+1$-order WL tree originate from the node q. This suggests that f_{q} maps $T^{(l)}_{\mathrm{q}_i}$ into a tree $T^{(l)}_{f(\mathrm{q}_i)} = T^{(l)}_{\mathrm{t}_i}$, which is a subtree of $T^{(l+1)}_{\mathrm{t}}$,. Then the condition 2) is true.
On the other hand, if condition 2) holds, then we define the mapping as $f_{\mathrm{q}}(v) = \begin{cases} f_{\mathrm{q}_i}(v), & v \in T^{(l)}_{\mathrm{q}_i} \\ \mathrm{q}, & v = \mathrm{q} \end{cases}$. Here, the function $f_{\mathrm{q}}(v)$ is a well-defined injective function because all $T^{(l)}_{\mathrm{q}_i}$ has no intersection. This implies this is a subtree isomorphic mapping, so 1) holds.

□

The above theorem provides a recursive solution to the WL subtree isomorphism algorithm. Intuitively, we can maintain an indicator matrix $S^{(l)} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$, where $S^{(l)}_{\mathrm{tq}} = \begin{cases} 1, & T^{(l)}_{\mathrm{q}} \subset T^{(l)}_{\mathrm{t}} \\ 0, & \text{else} \end{cases}$. This matrix captures the relation between all pairs of nodes and thus can be used for recursion update. Next, we will show that the update process can be implemented as a perfect matching problem, i.e., what makes condition 2) true is equivalent to finding a perfect matching on a bipartite graph, as shown in the following theorem:

Theorem A.4. *Assume the neighboring set of node* t *and* q *as $X = N(\mathrm{t})$ and $Y = N(\mathrm{q})$, respectively. Accordingly, we form a bipartite graph as $B^{(l)}_{\mathrm{t,q}}(X, Y, E)$. Here, we define the edges as $E = \{(\mathrm{t}_i, \mathrm{q}_j) : T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_j}\}$, where* t_i *and* q_j *represent the ith and jth neighbour of node* t *and* q*, respectively. Under this setting, the injective function f from the condition 2) in Theorem. A.3 induces a perfect matching.*

Proof. The injective function f of condition 2) in Theorem A.3 maps every node q_i in $N(\mathrm{q})$ to $\mathrm{t}_i = f(\mathrm{q}_i) \in N(\mathrm{t})$ and $T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_i}$ holds. While $T^{(l)}_{\mathrm{q}_i} \subset T^{(l)}_{\mathrm{t}_i}$ means $(\mathrm{q}_i, \mathrm{t}_i) \in E$, the injective f naturally corresponds every node q_i to an edge $(\mathrm{q}_i, \mathrm{t}_j)$. Since f is an injective function, $\mathrm{q}_{i_1} \neq \mathrm{q}_{i_2} \Rightarrow \mathrm{t}_{i_1} \neq \mathrm{t}_{i_2}$, indicating that all these edges $(\mathrm{q}_i, \mathrm{t}_i), i = 1, ..., |N(\mathrm{q})|$ are different, which actually forms a perfect matching. □

Theorem A.5. *Given the sampled query graph and the target graph, we can construct their adjacency matrices , $\tilde{A}_{\mathcal{Q}}$ and $A_{\mathcal{T}}$, and the degree matrix of the sampled query graph $\tilde{D}_{\mathcal{Q}} = diag(\sum_s((\tilde{A}_{\mathcal{Q}})_{:s}))$. Here, we denote the indicator matrix at the l-th hop as $S^{(l)}$. To check the validity of $|N(W)| \geq |W|$, we can check whether each element of Φ is true or not, where $\Phi := Z_{N(W)} \geq 1$, $Z_{N(W)} = AGG_{sum}(A_{\mathcal{T}}, Z^T_W)$ and $Z_W = AGG_{\max}(\tilde{D}^{-1}_{\mathcal{Q}} \cdot \tilde{A}_{\mathcal{Q}}, (S^{(l)})^T)$.*

Proof. For each node pair t, q and their corresponding $W = N'(\mathrm{q})$ in the sampled query graph, We first transform the neighboring set of W, i.e., $N(W)$, as following:

$$\begin{aligned} N(W) &= \{\mathrm{t}_i \in N(\mathrm{t}) | \exists \mathrm{q}_j \in W = N'(\mathrm{q}), \text{s.t.} T^{(l)}_{\mathrm{q}_j} \subset T^{(l)}_{\mathrm{t}_i}\} \\ &= \{\mathrm{t}_i \in N(\mathrm{t}) | \exists \mathrm{q}_j \in W = N'(\mathrm{q}), \text{s.t.} S_{\mathrm{t}_i,\mathrm{q}_j} = 1\} \\ &= \{\mathrm{t}_i \in N(\mathrm{t}) | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\} \\ &= N(\mathrm{t}) \cap \{\mathrm{t}_i | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\} \\ &= N(\mathrm{t}) \cap M(\mathrm{q}) \end{aligned} \tag{14}$$

Let $M(\mathrm{q}) = \{\mathrm{t}_i | \max_{\mathrm{q}' \in N'(\mathrm{q})} S^{(l)}_{\mathrm{t}_i,\mathrm{q}'} = 1\}$, we can compute $M(\mathrm{q})$ via a standard maximizing aggregation process on the sampled adjacency matrix $\tilde{A}_{\mathcal{Q}}$, in which treats the indicator matrix $(S^{(l)})^T \in R^{|V_{\mathcal{Q}}| \times |V_{\mathcal{T}}|}$ as node attributes. This process will output the representation of node q as follows,

$$z_{\mathrm{q},:} = max\{(S^{(l)})^T_{j,:}, \forall j \in N'(\mathrm{q})\}, \tag{15}$$

The obtained vector $z_{\mathrm{q},:}$ is to represent the neighbours of node q, i.e., $M(\mathrm{q})$, where $z_{\mathrm{q}i} = \begin{cases} 1, & i \in M(\mathrm{q}) \\ 0, & \text{else} \end{cases}$. We rewrite this into a matrix format as

$$Z_W = \text{AGG}_{\max}(\tilde{A}_{\mathcal{Q}}, (S^{(l)})^T) \tag{16}$$

where $Z_W \in R^{|V_{\mathcal{Q}}| \times |V_{\mathcal{T}}|}$ and its q-th row vector is $z_{\mathrm{q}:}$.

Recall that we demand $N(W) = N(\mathrm{t}) \cap M(\mathrm{q})$. After acquiring $M(\mathrm{q})$, we can compute the $|N(W)|$ as follows,

$$\begin{cases} |M(\mathrm{q})| = \sum_i z_{\mathrm{q},i} \\ |N(W)| = \sum_i z_{\mathrm{q},i}, i \in N(\mathrm{t}) \end{cases} \tag{17}$$

In essence, this is to implement a summation aggregation on the target graph using the node representation Z_W, i.e.,

$$Z_{N(W)} = \text{AGG}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T) \tag{18}$$

where $Z_{N(W)} \in R^{|V_{\mathcal{T}}| \times |V_{\mathcal{Q}}|}$ is an integer matrix and its element (t, q) shows the score of $|N(W)|$ between node t and q. This transformation converts the counting operation as aggregation such that we can check the aggregated values to determine whether there is a perfect matching. Given a node pair (t, q), we have $|N(W)| = [Z_{N(W)}]_{\mathrm{tq}}$ and $|W| = |N'(\mathrm{q})| = \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s}$. Therefore, the question becomes to check whether $[Z_{N(W)}]_{\mathrm{tq}} \geq \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s}$ holds. We can then derive the perfect matching as follows:

$$\begin{aligned} &[Z_{N(W)}]_{\mathrm{tq}} \geq \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s} \\ \Leftrightarrow &[Z_{N(W)}]_{\mathrm{tq}} / \sum_s [\tilde{A}_{\mathcal{Q}}]_{\mathrm{q}s} \geq 1 \\ \Leftrightarrow &[\text{AGG}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T)]_{\mathrm{tq}} / \tilde{d}_{\mathrm{q}} \geq 1 \\ \Leftrightarrow &[\text{AGG}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T) \cdot \tilde{D}_{\mathcal{Q}}^{-1}]_{\mathrm{tq}} \geq 1 \\ \Leftrightarrow &[\text{AGG}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T \cdot \tilde{D}_{\mathcal{Q}}^{-1})]_{\mathrm{tq}} \geq 1 \end{aligned} \tag{19}$$

where $\tilde{d}_{\mathrm{q}}$ is the degree of node q in the sampled graph. The degree matrix of the sample graph is defined as $\tilde{D}_{\mathcal{Q}} = \text{diag}[\sum_s ((\tilde{A}_{\mathcal{Q}})_{:s})]$. Now recall that Φ is the matrix whose (t, q) element is the comparison result of $|N(W)|$ and $|W|$ of (t, q), according to eq 19, we have:

$$\Phi^{(l+1)}(\tilde{A}_{\mathcal{Q}}, A_{\mathcal{T}}) = \text{AGG}_{\text{sum}}(A_{\mathcal{T}}, Z_W^T \cdot \tilde{D}_{\mathcal{Q}}^{-1}) \geq 1, \tag{20}$$

where

$$\begin{aligned} Z_W^T \cdot \tilde{D}_Q^{-1} &= [\text{AGG}_{\max}(\tilde{A}_Q, (S^{(l)}))]^T \cdot \tilde{D}_Q^{-1} \\ &= [\tilde{D}_Q^{-1} \cdot \text{AGG}_{\max}(\tilde{A}_Q, (S^{(l)}))]^T \\ &= [\text{AGG}_{\max}(\tilde{D}_Q^{-1} \cdot \tilde{A}_Q, (S^{(l)}))]^T \end{aligned} \tag{21}$$

□

Theorem A.6. *Every chordless cycle is atomic. Every chordless cycle $\mathcal{C}_Q$ in an induced subgraph G_Q must correspond to a chordless cycle $\mathcal{C}_\mathcal{T}$ in the origin graph $G_\mathcal{T}$.*

Proof. Chordless cycle does not have any chord, thus there is no smaller cycle in the chordless cycle, which means chordless cycle is atomic. Assuming G_Q is a subgraph of $G_\mathcal{T}$, every node of $\mathcal{C}_Q$ must correspond to a node in $G_\mathcal{T}$, and these nodes form a circle $\mathcal{C}_\mathcal{T}$ in $G_\mathcal{T}$. Since G_Q is an induced subgraph of $G_\mathcal{T}$, if $\mathcal{C}_\mathcal{T}$ has a chord, then $\mathcal{C}_Q$ must have a chord, which contradicts the condition that $\mathcal{C}_Q$ is a chordless graph. □

A.3. Implementation Details

In this section we present the implementation details of our D^2Match[1]. At the beginning of subtree isomorphism test, the model needs an initial indicator matrix $S_{subtree}^{(0)}$ as the input of the first iteration. According to the definition of the indicator matrix, $S_{subtree}^{(0)}$ shows the isomorphism relation between the subtree of 0-hop neighbors, which are the nodes themselves in this case. Since all nodes will be isomorphic to each other if not considering the node attributes, the indicator matrix $S_{subtree}^{(0)}$ is actually a similarity matrix w.r.t node attributes. To get a similarity matrix of attributes, we can either directly calculate the similarity between nodes or employ neural networks on these attributes to learn the matrix. In our model, we implement both methods to initialize the matrix, called the initialization of the raw and the learnable:

$$\begin{aligned} Raw: S_{subtree}^{(0)} &= CosineSimilarity(X_\mathcal{T}, X_Q) = Norm(X_\mathcal{T}) \cdot Norm(X_Q^T) \\ Learnable: S_{subtree}^{(0)} &= MLP(X_\mathcal{T}) \cdot MLP(X_Q)^T \end{aligned} \tag{22}$$

where the raw initialization is to calculate the cosine similarity between the nodes' attributes, and the learnable initialization employs a MLP to generate hidden representations of nodes and compute their dot similarities.

In practice, we find the raw initialization performs better. This is because the node attributes of datasets are usually binary categorical vectors, which induces clear identification information of the nodes and can be easily captured by cosine similarity.

Our implementation of the GNN block in the model is slightly different from the description. Specifically, we use compute the similarity of each pair of nodes as:

$$[S_{gnn}^{(l+1)}]_{ij} = MLP(concat([H_\mathcal{T}^{(l)}]_i, [H_Q^{(l)}]_j)). \tag{23}$$

The main difference is that we do not output a $|V_\mathcal{T}| \times |V_Q|$ matrix, but a $|V_\mathcal{T}| \times |V_Q| \times |D^{(l+1)}|$ tensor, where $D^{(l+1)}$ denotes the hidden dim of $l+1$ layer. The intuition is that a tensor that represents the node pairs' similarity with vectors can retain more information than a similarity matrix with scalar elements. In this setting, the final indicator matrix $S^{(l+1)}$ can not be generated as $S^{(l+1)} = S_{gnn}^{(l+1)} \odot S_{subtree}^{(l+1)}$, because $S_{subtree}^{(l+1)} \in R^{|V_\mathcal{T}| \times |V_Q|}$ but $S_{gnn}^{(l+1)} \in R^{|V_\mathcal{T}| \times |V_Q| \times |D^{(l+1)}|}$. Thus we broadcast $S_{subtree}^{(l+1)}$ to $\tilde{S}_{subtree}^{(l+1)}$ where $\forall k \in [0, D^{(l+1)})$, $[\tilde{S}_{subtree}^{(l+1)}]_{ijk} = [S_{subtree}^{(l+1)}]_{ij}$ and the final indicator matrix $S^{(l+1)} = S_{gnn}^{(l+1)} \odot \tilde{S}_{subtree}^{(l+1)}$

At the end of our models, we get the subtree indicator matrix $S_{subtree}^{(L)}$ and the GNN indicator matrix $S_{gnn}^{(L)}$. The model will output the final score from $S_{subtree}^{(L)}$ and $S_{gnn}^{(L)}$, respectively. For the subtree module, we check whether the indicator matrix is feasible to induce the subgraph isomorphism. Note that for a node i in the target graph and a node j in the query graph, i is possible to match j unless $[S_{subtree}^{(L)}]_{ij} = 1$. So we check whether the subtree indicator matrix meets the following two conditions:

[1]The python implementation of D^2Match will be available at https://github.com/XuanzhouLiu/D2Match-ICML23

1) Every node in a query graph should match at least one node in the target graph:

$$\begin{aligned} &\forall j, \max_i (S^{(L)}_{subtree})_{ij} = 1 \\ &\Leftrightarrow \sum_j \max_i (S^{(L)}_{subtree})_{ij} = |V_\mathcal{Q}| \\ &\Leftrightarrow \sum_j \max_i (S^{(L)}_{subtree})_{ij} / |V_\mathcal{Q}| = 1 \end{aligned} \tag{24}$$

2) The number of nodes in the target graph that match at least one node in the query graph is more than the number of nodes of query graph:

$$\sum_i \max_j (S^{(L)}_{subtree})_{ij} \geq |V_\mathcal{Q}| \Leftrightarrow \quad \sum_i \max_j (S^{(L)}_{subtree})_{ij} / |V_\mathcal{Q}| \geq 1 \tag{25}$$

To make the subtree model differentiable, we use a sigmoid to replace all the logical judgment in the model:

$$\sigma(x) = \text{Sigmoid}(ax + b) \tag{26}$$

where a, b are learnable parameters; σ is the sigmoid function. The result of subtree module can be fomulated as:

$$r_{subtree} = \sigma(\sum_i \max_j (S^{(L)}_{subtree})_{ij} / |V_\mathcal{Q}|) \cdot \sigma(\sum_j \max_i (S^{(L)}_{subtree})_{ij} / |V_\mathcal{Q}|) \tag{27}$$

For the GNN module, we employ the neural tensor network(NTN) (Bai et al., 2019) and generate a score according to the output of NTN and the aggregated indicator tensor:

$$r_{gnn} = \sigma(MLP(concat[NTN(H^{(L)}_\mathcal{T}, H^{(L)}_\mathcal{Q}), \sum_i \sum_j S^{(L)}_{subtree}])) \tag{28}$$

Where $H^{(L)}_\mathcal{T}, H^{(L)}_\mathcal{Q}$ are the node representations generated by the GNNs. NTN is the NTN layer.

The final prediction is:

$$r = r_{gnn} \cdot r_{subtree} \tag{29}$$

Although the model's prediction is obtained by integrating the two modules, we can not directly train the model through the final score r because it will bring difficulties in the training process. When fitting a negative sample, the resulting subtree module tends to be zero, forcing the overall gradient to be zero which hinders the training of the GNN module.

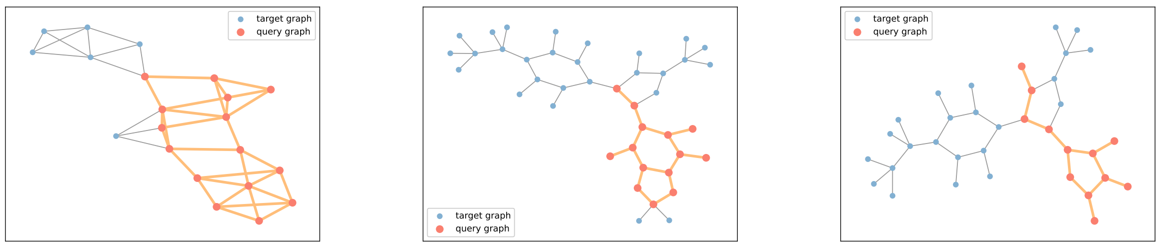
Therefore, we train the two blocks with different objectives. For the subtree module which aims to learn the isomorphism relation, the result should be either 0 for not matching or 1 for matching. So we employ MAE loss to enforce the results to be either 0 or 1. For the GNN module, we use MSE to encourage the output of GNNs to capture the similarity. Suppose the ground-truth label is y, and our loss function is

$$L = MSE(r_{gnn}, y) + MAE(r_{subtree}, y) \tag{30}$$

Both our model and all baselines use the Adam as optimizer and set the learning rate to $3e - 4$. To ensure fairness, we set all models with adjustable number of layers to 5 layers, and set the hidden dimension to 10 to avoid overfitting.

A.4. D^2Match at Work

Recall that D^2Match learns an indicator matrix to capture pairwise similarities. It plays the role of permutation matrix in matching, allowing us to pinpoint the matched subgraph. This is particularly useful since the exact position is required for some downstream applications such as web search. In comparison, other learning-based methods are unable to pinpoint local correspondences, but only establish the existence of a matching. We provide a visualization of the matched subgraph to better understand the problem difficulty and the effectiveness of our method, as shown in the Fig. 3.



(a) (b) (c)

Figure 3. The detected subgraphs by D^2Match

Table 7. The ablation study of D2Match module

	Synthetic	Synthetic$^+$	Proteins	Proteins*	IMDB-Binary	IMDB-Binary*	FirstMMDB	FirstMMDB*
D^2Match (gnn only)	$61.1_{\pm 13.31}$	$70.2_{\pm 18.58}$	$95.2_{\pm 1.04}$	$77.2_{\pm 8.11}$	$50.0_{\pm 0.00}$	$64.4_{\pm 19.73}$	$69.7_{\pm 26.98}$	$67.8_{\pm 24.38}$
D^2Match (subtree only)	$70.0_{\pm 2.09}$	$74.8_{\pm 2.56}$	$100.0_{\pm 0.00}$	$82.0_{\pm 2.92}$	$92.9_{\pm 1.04}$	$82.8_{\pm 4.02}$	$100.0_{\pm 0.0}$	$72.0_{\pm 6.20}$
D^2Match	$72.7_{\pm 4.45}$	$86.6_{\pm 1.44}$	$100.0_{\pm 0.00}$	$83.4_{\pm 2.97}$	$93.3_{\pm 1.03}$	$90.2_{\pm 1.79}$	$100.0_{\pm 0.0}$	$86.4_{\pm 7.44}$

A.5. Ablation Studies

We perform ablation studies for the GNN module, subtree module, and chordless cycle.

The GNN module in our analysis will capture the distributional features on the graph, such as the edge density difference between classes. The GNN module is thus essential for datasets with multiple distributions, also called biased data. We run experiments on both the biased and unbiased synthetic datasets to show the performance of our method and its variation that without the GNN module, as shown in Table 7. D^2Match outperforms D^2Match without the GNN module as our theory predicts. But our GNN module shares the same weaknesses as the other GNN models when dealing with evenly distributed data. We observe that the performance of the GNN module drops significantly on hard datasets similar to other GNN models. The subtree module can significantly improve the performance because it harnesses the property of subgraph-matched data, making it robust to data's distribution. Our subtree module outperformed the GNN module on all datasets in our ablation study, demonstrating its effectiveness.

We also perform the ablation study on the Synthetic dataset to test the effect of chordless cycles,as shown in Table 8. Results show the chordless cycles boost the performance with limited extra time consumption.

Table 8. The ablation study of cc

	Synthetic	RunTime
D^2Match	$74.3_{\pm 1.60}$	19.7s/epoch
D^2Match (w/o cc)	$72.7_{\pm 4.45}$	10.3s/epoch

Table 9. Random seed comparison

	proteins	mutag
Seed(0)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Seed(1)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Seed(2)	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$
Fixed	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$

Table 10. Runtime analysis

	Training(s/epoch)	Inference(s/epoch)
SimGNN	1.732	0.385
NeuroMatch	2.234	0.311
GMN-embed	1.850	0.290
GraphSim	3.223	0.433
IsoNet	10.553	1.939
D^2Match-Subtree(S=2)	2.940	0.456
D^2Match-Subtree(S=3)	3.889	0.581
D^2Match-Subtree(S=4)	4.410	0.673
D^2Match-Subtree(S=5)	5.143	0.750
D^2Match-GNN	2.678	0.495
D^2Match	8.163	1.114

A.6. Random Effect

Although our experiments do not rely on random seeds, a random split may affect the results. To test this, we set up several random seeds and permute the raw data order before getting the five-fold. We experiment on the Protein and Mutag datasets with trivial random seed 0,1,2 and obtain nearly identical performance. See Table 9.

While other methods based on GNNs tend to capture the divergence of distributions in the training set and hence are easily affected by randomness, our subtree module performs the matching explicitly by the degeneracy property, as opposed to modeling the data distribution in others, hence ours is insensitive to data partitioning.

Table 11. The hard dataset details

	Synthetic^{+}	Proteins*	Mutag*	Enzymes*	Aids*	IMDB-Binary*	Cox2*	FirstMMDB*
Average nodes (target)	40.0	38.8	18.2	31.5	14.7	19.0	41.3	1376.7
Average nodes (query)	15.0	11.4	9.1	15.4	4.4	14.2	15.0	15.0
Average edges (target)	259.5	146.7	40.2	120.6	30.0	177.1	87.0	6141.6
Average edges (query)	67.3	35.5	17.6	52.6	7.1	102.6	29.9	45.6

Table 12. The dataset details

	Synthetic	Proteins	Mutag	Enzymes	Aids	IMDB-Binary	Cox2	FirstMMDB
Average nodes (target)	40.0	39.1	17.9	33.0	15.7	19.8	41.3	1376.5
Average nodes (query)	15.0	14.4	9.0	14.8	7.9	14.6	14.4	15.0
Average edges (target)	241.7	146.5	39.5	125.6	32.4	193.1	87.0	6144.3
Average edges (query)	50.6	68.9	25.1	75.3	17.1	141.0	42.8	68.1

A.7. Runtime Analysis

We add the runtime analysis experiment as follows. We compare our method with baselines on the synthetic dataset and record the training and inference time (second) per epoch. The results are shown in Table 10.

Our model is slower than some strong baselines like SimGNN and NeuroMatch in the experiment because they deal with the graph-level representations. Our model is faster than IsoNet, which performs edge-level matching.

We conduct an additional ablation study to explore the time consumption of each module in our model. The results show that the time consumption of our model mainly comes from the sampling in the subtree module whose running time is linearly related to the sampling number. When we set the sampling number as 2, the running time is on par with the others. Furthermore, the running time for the GNN module is the same as for the other baselines. In sum, we observe that our model's scalability is acceptable as both complexity analysis and empirical running time show ours is slower than others only by a constant factor.

A.8. Dataset Details

We describe the average node number and average edge number of the target graph and query graph in the Table 12 and Table 11. Except the hard datasets, we generate 1000 graph pairs for Synthetic, Proteins, Mutag, Enzymes, Cox2 and FirstMMDB and 2000 graph pairs for Aids and IMDB-Binary which have smaller graph size. For the hard dataset, we uniformly generate 500 graph pairs.

A.9. Results on More Datasets

We conduct experiments on the OGB benchmark dataset (Hu et al., 2020), including Ogbg-molhiv and Ogbg-molpcb. We follow the same strategy in the paper to construct normal and hard versions for these datasets and choose the best-performing baselines for comparison, including SimGNN and NeuroMatch. We present new results in Table 13.

We find that our model performs slightly better than others on normal datasets while gaining a significant advantage over baselines on hard datasets. These results are consistent with our previous experiments, demonstrating that our model exploits the subgraph matching property, rather than simply modeling the divergence of the data distribution as other GNNs.

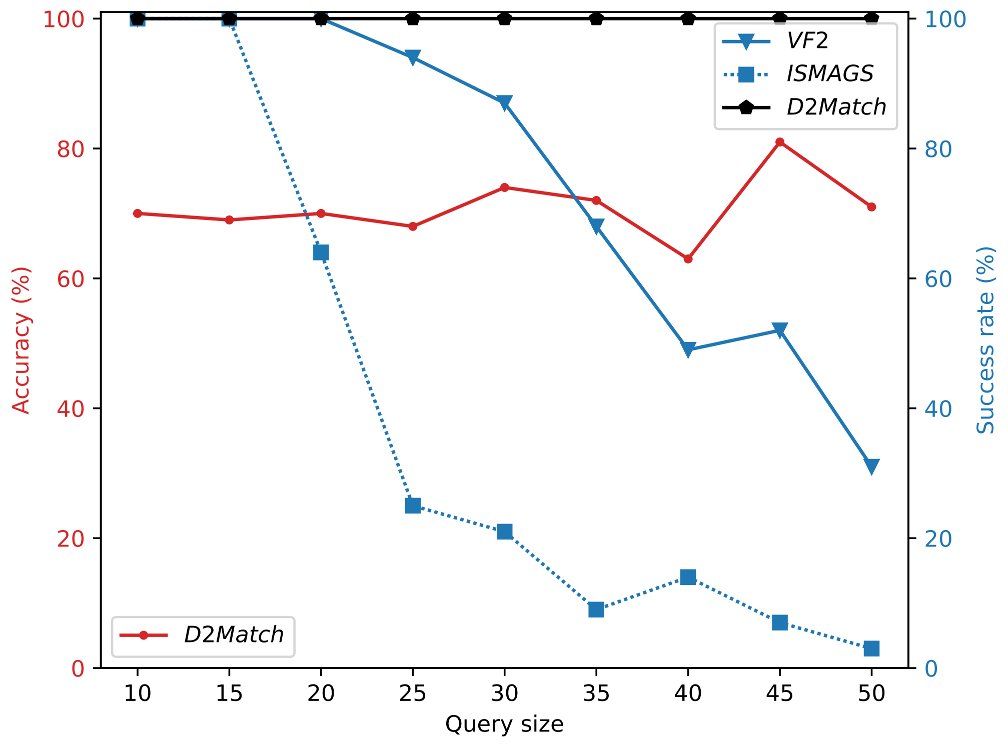


Figure 4. Comparison with exact methods

We experiment on continuous features from the MNIST, CIFAR10 and PPI datasets, as these are constructed from vision data(Dwivedi et al., 2020) or biological information data(Zitnik & Leskovec, 2017). We As expected, our model achieves consistent performance as well. See Table 14.

Table 13. Obg dataset performance comparison

	ogb-molhiv	ogb-molhiv*	ogb-molpcba	ogb-molpcba*
SimGNN	$99.4_{\pm 0.65}$	$81.6_{\pm 2.70}$	$99.8_{\pm 0.27}$	$86.2_{\pm 2.28}$
NeuroMatch	$98.3_{\pm 1.68}$	$86.0_{\pm 3.54}$	$99.8_{\pm 0.27}$	$90.6_{\pm 3.51}$
D^2Match	$99.8_{\pm 0.27}$	$99.6_{\pm 0.54}$	$100.0_{\pm 0.00}$	$100.0_{\pm 0.00}$

Table 14. Continues dataset performance

	Cifar10	MNIST	PPI
SimGNN	$89.0_{\pm 21.82}$	$98.5_{\pm 0.93}$	$77.0_{\pm 24.67}$
NeuroMatch	$98.1_{\pm 1.14}$	$95.9_{\pm 1.34}$	$50.0_{\pm 0.00}$
D^2Match	$99.3_{\pm 0.27}$	$98.8_{\pm 1.15}$	$98.8_{\pm 1.06}$

A.10. Comparison with Exact Methods

we compare exact matching solutions, including VF2 and ISMAGS. By nature, we know that all the exact matching methods can obtain 100 % accuracy.

As a trade-off between accuracy and execution time, we make the comparison inspired by the setup in NeuroMatch (Rex et al., 2020). We say an execution succeeds when its run time is less than 60s. We compare the success rate of the exact methods by varying the query graph size from 10 to 50 on the synthetic data and report the results in Figure 4.

We show in our experiment that the failure of exact matching methods increases significantly when the target graph has more than 30 nodes, compared to our stable performance, indicating the incompetence of these methods on large-scale datasets.