GREED: A Neural Framework for Learning Graph Distance Functions

Anonymous Author(s)
Affiliation
Address
email

Abstract

Among various distance functions for graphs, graph and subgraph edit distances (GED and SED respectively) are two of the most popular and expressive measures. Unfortunately, exact computations for both are NP-hard. To overcome this computational bottleneck, neural approaches to learn and predict edit distance in polynomial time have received much interest. While considerable progress has been made, there exist limitations that need to be addressed. First, the efficacy of an approximate distance function lies not only in its approximation accuracy, but also in the preservation of its properties. To elaborate, although GED is a metric, its neural approximations do not provide such a guarantee. This prohibits their usage in higher order tasks that rely on metric distance functions, such as clustering or indexing. Second, several existing frameworks for GED do not extend to SED due to SED being asymmetric. In this work, we design a novel siamese graph neural network called GREED, which through a carefully crafted inductive bias, learns GED and SED in a property-preserving manner. Through extensive experiments across 10 real graph datasets containing up to 7 million edges, we establish that GREED is not only more accurate than the state of the art, but also up to 3 orders of magnitude faster. Even more significantly, due to preserving the triangle inequality, the generated embeddings are indexable and consequently, even in a CPU-only environment, GREED is up to 50 times faster than GPU-powered baselines.

1 Introduction and Related Work

A distance function on any dataset, including graphs, is a fundamental operator. Among several distance measures on graphs, *edit distance* is one of the most powerful and popular mechanisms [28, 43, 41, 19]. Edit distance can be posed in two forms: *graph edit distance* (GED) and *subgraph edit distance* (SED). Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, GED$(\mathcal{G}_1, \mathcal{G}_2)$ returns the minimum cost of *edits* needed to convert $\mathcal{G}_1$ to $\mathcal{G}_2$, i.e., for $\mathcal{G}_1$ to become *isomorphic* to $\mathcal{G}_2$. An edit can be the addition or deletion of edges and nodes, or the replacement of edge or node labels, with an associated cost. In SED$(\mathcal{G}_1, \mathcal{G}_2)$, the goal is to identify the minimum cost of edits so that $\mathcal{G}_1$ is a subgraph (*subgraph isomorphic*) of $\mathcal{G}_2$. For examples, see Fig. 1.

GED is typically restricted to graph databases containing small graphs to facilitate distance computation with queries of similar sizes. As an example, given a repository of molecules, and a query molecule, we may want to identify the closest molecule in the repository that is similar to the query. SED, on the other hand, is useful when the database has large graphs and the query is a comparatively smaller graph. As examples, subgraph queries are used on knowledge graphs for analogy reasoning [16]. In PPI and chemical compounds, SED is of central importance to identify functional motifs and binding pockets [34, 19, 15]. Unfortunately, both GED and SED are NP-hard to compute [41, 19]. To mitigate this computational bottleneck, several heuristics [9, 12] and index structures [19, 41, 28, 43] have been proposed. Recently, graph neural networks have been shown to be effective in learning and predicting GED [2, 37, 27, 3, 42, 38, 13, 1]. The basic goal in all these algorithms is to learn a neural model from a training set of graph pairs and their distances, such that, at inference time, given an unseen graph pair, we are able to predict its distance accurately. In the

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

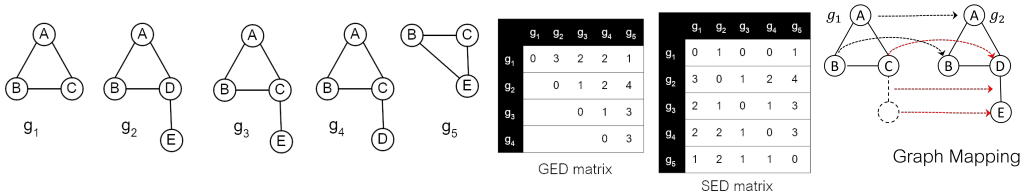


Figure 1: **A sample set of graphs (g_1-g_5), their corresponding GED and SED matrices and an example of a graph mapping from g_1 to g_2. The dashed nodes and edges in the mapping represent dummy nodes and edges. The red arrows denote either insertion or change of label.**

domain of subgraphs, NEUROMATCH [32] and ISONET [33] generate embeddings to detect subgraph isomorphism. NSC [36] generates subgraph level embeddings that can count the number of subgraph instances of a query graph on a target graph. While the progress made is impressive, there is scope to do more.

- **Preservation of theoretical properties:** GED is a *metric* distance function. While SED is not metric due to being asymmetric, it satisfies the *triangle inequality*, *non-negativity*, and *subgraph-identity*. Several higher-order tasks such as clustering and indexing rely on such properties [18, 14, 20, 35, 11]. Existing neural approaches do not preserve these properties, which limits their usability for these higher order tasks.
- **Indexable embeddings:** Given graph pair $\mathcal{G}_1$ and $\mathcal{G}_2$, neural approaches first embed them into a feature space. Next, they compute a distance on these feature vectors, which is an approximation of the distance in the original graph space. The literature on indexing range and k-NN queries over feature vectors is rich [17, 23, 11]. Index structures typically allow sub-linear computation costs with respect to the database size. Unfortunately, none of the neural approaches generate indexable feature vectors since they perform *pair-dependent* computations. Specifically, the neural computations on $\mathcal{G}_1$ depend on both $\mathcal{G}_1$ and $\mathcal{G}_2$ (and same for $\mathcal{G}_2$). Consequently, the computations can only be done at query-time and thereby negating the possibility of indexing pre-computed feature space embeddings.
- **Modeling SED:** Existing neural approaches to learning GED cannot easily be adapted to learn SED. While GED is symmetric, SED is not. Several neural architectures for GED have the assumption of symmetry at its core and hence modeling SED is non-trivial [2, 3, 27].
- **Exponential Search Space:** Computing $\text{SED}(\mathcal{G}_1, \mathcal{G}_2)$ conceptually requires us to compare the query graph $\mathcal{G}_1$ with the exponentially many subgraphs of the target graph $\mathcal{G}_2$. Therefore, it is imperative that the model has an efficient and effective mechanism to prune the search space without compromising on the prediction accuracy.

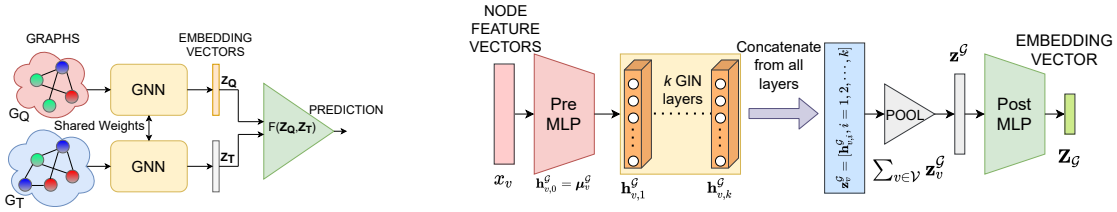
In this work, we address the above limitations through the following contributions.

- **Novel neural architecture:** We address the above mentioned challenges through a novel architecture called GREED: GRaph Embeddings for Edit Distances. GREED utilizes a *siamese graph isomorphism network* [39] to embed graphs in a *pair-independent* fashion. A simple, but theoretically well-characterized, function on this embedding space predicts the SED and GED. The carefully crafted prediction function serves as an *inductive bias* for the model, which, in addition to enabling high generalization accuracy, preserves the metric property of GED and the triangle inequality of SED in the embedding space.
- **Indexable embeddings:** Owing to pair-independent embeddings and preservation of the triangle inequality over the embedding space for both SED and GED, GREED can exploit the rich literature on index structures [17, 23, 11] to boost efficiency.
- **Accurate, Fast and Scalable:** Extensive experiments on real graph datasets containing up to a million nodes establish that GREED is more accurate in both GED and SED when compared to the state of the art algorithms and is more than 3 orders of magnitude faster in range and k-NN queries. Furthermore, owing to indexable embeddings, even in a CPU-only environment, GREED is up to 50 times faster than the closest baseline run on a GPU.

2 Preliminaries and Problem Formulation

We denote a labeled undirected graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set and $\mathcal{L} : \mathcal{V} \cup \mathcal{E} \rightarrow \Sigma$ is the labeling function over nodes and edges. Σ is the universe of all labels and contains a special empty label ϵ. $\mathcal{L}(v)$ and $\mathcal{L}(e)$ denote the labels of node v and edge e respectively. $\mathcal{G}_1 \subseteq \mathcal{G}_2$ denotes that $\mathcal{G}_1$ is a *subgraph* of $\mathcal{G}_2$.

The problem of learning GED and SED [19] is defined as follows.



(a) Siamese architecture of GREED

(b) The GNN component in GREED.

Figure 2: **The architecture of GREED.**

Problem 1 (Learning GED/SED) *Given a training set of tuples of the form* $\langle \mathcal{G}_1, \mathcal{G}_2, \text{GED}(\mathcal{G}_1, \mathcal{G}_2)\rangle$ *(or* $\mathcal{G}_1, \mathcal{G}_2, \text{SED}(\mathcal{G}_1, \mathcal{G}_2)\rangle$*), learn a neural model to predict* $\text{GED}(\mathcal{Q}_1, \mathcal{Q}_2)$ *(or* $\text{SED}(\mathcal{Q}_1, \mathcal{Q}_2)$*) on unseen graphs* $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$*.*

For details on the exact definition of GED and SED, we refer to the appendix (App. A).

2.1 Properties of GED and SED

Theorem 1 *Let* $\widehat{d} : \Sigma \times \Sigma \rightarrow \mathbb{R}_0^+$ *be a distance function over* Σ*, where (i)* $\widehat{d}(\ell_1, \ell_2) = 0$ *if* $\ell_1 = \epsilon$*, and (ii)* $\widehat{d}(\ell_1, \ell_2) = d(\ell_1, \ell_2)$ *otherwise; the following holds:* $\text{SED}(\mathcal{G}_1, \mathcal{G}_2) = \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$*, where* $\widehat{\text{GED}}$ *denotes* GED *with* $\widehat{d}$ *as the label set distance function. In simple words, the* SED *between two graphs is equivalent to* GED *with a label set distance function where we ignore insertion costs.*

PROOF. See App. B.1.

Observation 1 GED *satisfies the triangle inequality if the distance function* d *over label set* Σ *satisfies the triangle inequality [19]. As defined in the paragraph following Def. 2,* d *satisfies the triangle inequality [19]. Furthermore, it is trivial to see that* GED *is symmetric, non-negative and satisfies identity as long as* d *satisfies the analogues. Hence,* GED *with distance function* d *is metric.*

Theorem 2 SED *is not metric due to violating the properties of symmetry and identity. However, it satisfies the triangle inequality, i.e.,* $\text{SED}(\mathcal{G}_1, \mathcal{G}_3) \leq \text{SED}(\mathcal{G}_1, \mathcal{G}_2) + \text{SED}(\mathcal{G}_2, \mathcal{G}_3)$*.*

PROOF: See App. B.2 for details.

Observation 2 *Computing* GED *and* SED *is NP-hard [41].*

Hereon, we use GED as the illustrative distance function being modeled. The architecture trivially extends to SED. The specific places that need separate treatment will be discussed explicitly.

3 GREED: The Proposed Architecture

Fig. 2 presents the architecture of GREED. The input to our learning framework is a pair of graphs $\mathcal{G}_\mathcal{Q}$ (query), $\mathcal{G}_\mathcal{T}$ (target) along with the supervision data $\text{GED}(\mathcal{G}_Q, \mathcal{G}_T)$ (or $\text{SED}(\mathcal{G}_Q, \mathcal{G}_T)$). Our objective is to train a model that can predict GED on unseen query and target graphs. The design of our model must be cognizant of the fact that computing GED is NP-hard and high quality training data is scarce. Thus, we use a *Siamese* architecture [10], where there are *two* networks with *shared* parameters applied to two inputs independently to compute representations.

3.1 Siamese Graph Neural Network

As depicted in Fig. 2a, we use a siamese graph neural network (GNN) with shared parameters to embed both $\mathcal{G}_\mathcal{Q}$ and $\mathcal{G}_\mathcal{T}$. While one could use two different GNN models for the query and the target, this design increases the model parameters and consequently, the training time. Furthermore, an architecture with higher number of parameters also requires larger amount of training data, which is difficult due to GED being NP-hard.

Fig. 2b focuses on the GNN component of GREED. We next discuss each of its individual components.

PRE-MLP: $\mathbf{x}_v$ in Fig. 2b is a one-hot encoding of the categorical node labels. The dimension of the one-hot vector increases linearly with the number of labels in a graph database and hence can be very large. The primary-job of the pre-mlp is to reduce it to a desirable dimension size through the operation $\boldsymbol{\mu}_v^\mathcal{G} = \text{MLP}(\mathbf{x}_v)$. Indeed, a similar effect may also be obtained by directly feeding the one-hot encoding to the first layer of GIN. However, since a GIN constructs embeddings by incorporating

both structure and label information, one may desire a different dimensionality in the GIN layers. Hence, the pre-mlp is motivated more from a conceptual separation of its task to that of GIN rather than a purely performance point of view (See App. G). We do not explicitly model edge labels in our experiments. GREED can easily be extended to edge labels by using GINE [21] instead of GIN.

Graph Isomorphism Network (GIN): GIN [39] consumes the information from the Pre-MLP to learn hidden representations that encode both the graph structure as well as the node feature information. GIN is as powerful as the *Weisfeiler-Lehman (WL) graph isomorphism test* [24] in distinguishing graph structures. Since our goal is to accurately characterize graph topology and learn similarity, GIN emerges as the natural choice. GIN develops its expressive power by using an *injective* aggregation function. Specifically, in the initial layer, each node v in graph $\mathcal{G}$ is characterized by the representation learned by the MLP, i.e., $\mathbf{h}_{v,0}^{\mathcal{G}} = \boldsymbol{\mu}_v^{\mathcal{G}}$. Subsequently, in each hidden layer i, we learn an embedding through the following transformation.

$$\mathbf{h}_{v,i}^{\mathcal{G}} = \text{MLP}\left((1+\epsilon^i)\cdot \mathbf{h}_{v,i-1}^{\mathcal{G}} + \sum_{u\in\mathcal{N}_{\mathcal{G}}(v)} \mathbf{h}_{u,i-1}^{\mathcal{G}}\right) \tag{1}$$

Here, ϵ^i is a layer-specific learnable parameter, $\mathcal{N}_{\mathcal{G}}(v)$ is one-hop neighbourhood of the node v, and $\mathbf{h}_{v,0}^{\mathcal{G}} = \boldsymbol{\mu}_v^{\mathcal{G}}$. The k-th layer embedding is $\mathbf{h}_{v,k}^{\mathcal{G}}$, where k is final hidden layer.

Concatenation, Pool and Post-MLP: Intuitively, $\mathbf{h}_{v,i}^{\mathcal{G}}$ captures a feature-space representation of the i-hop neighborhood of v. Typically, GNNs operate on node or edge level predictive tasks, such as node classification or link prediction, and hence, the node representations are passed through an MLP for the final prediction task. In our problem, we need to capture a graph level representation. Furthermore, the representation should be rich enough to also capture the various subgraphs within the input graph so that SED can be predicted accurately. To fulfil these requirements, we first *concatenate* the representation of a node across *all* hidden layers, i.e., the final node embedding is $\mathbf{z}_v^{\mathcal{G}} = \text{CONCAT}\left(\mathbf{h}_{v,i}^{\mathcal{G}}, \forall i \in \{1,2,\cdots,k\}\right)$. This allows us to capture a multi-granular view of the subgraphs centered on v at different radii in the range $[1,k]$. Next, to construct the graph-level representation, we perform a sum-pool, which adds the node representations to give a single vector. This information is then fed to the Post-MLP to enable post-processing. Mathematically:

$$\mathbf{Z}_{\mathcal{G}} = \text{MLP}(\mathbf{z}^{\mathcal{G}}) = \text{MLP}\left(\sum_{v\in\mathcal{V}} \mathbf{z}_v^{\mathcal{G}}\right) \tag{2}$$

GED and SED Prediction: The final task is to predict the GED (and SED) as a function of query graph embedding $\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}$ and target graph embedding $\mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}$. The natural choice would be to feed these embeddings into another MLP to learn $\text{GED}(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}})$. This MLP can then be trained jointly with the graph embedding model in an *end-to-end* fashion. However, an MLP prediction does not have any theoretical guarantees with respect to the preservation of metric properties of GED and the triangle inequality of SED. We, therefore, focus on learning prediction functions $\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right)$ and $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right)$ for GED and SED respectively, such that they are accurate and respects the desirable properties from the original graph space. As we will empirically substantiate in § 4.5, the inductive bias injected through the prediction functions also lead to more effective learning over low volumes of training data than an MLP.

3.1.1 GED

We require the following four properties to ensure that the prediction is also a metric.

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) \geq 0 \tag{3}$$

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) = 0 \iff \forall i : \mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}[i] = \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}[i] \tag{4}$$

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) = \mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}\right) \tag{5}$$

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) \leq \mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}'}\right) + \mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}'}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) \tag{6}$$

To achieve this, we establish an important connection of metrics on vector spaces to norms. Every norm $\|.\|$ gives a metric $(\mathbf{x},\mathbf{y}) \mapsto \|\mathbf{x}-\mathbf{y}\|$. Moreover for a metric, there exists a norm $\|.\|$ such that the metric can be expressed as $(\mathbf{x},\mathbf{y}) \mapsto \|\mathbf{x}-\mathbf{y}\|$, *iff* the metric is *translation invariant* and *homogeneous*. Thus, we add these properties to the desiderata for $\mathcal{F}_g$:

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}} + \mathbf{k}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}} + \mathbf{k}\right) = \mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right), \forall \mathbf{k} \in \mathbb{R}^d \tag{7}$$

$$\mathcal{F}_g\left(r\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, r\mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right) = |r|\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}}}\right), \forall r \in \mathbb{R} \tag{8}$$

Armed with these observations, we define the class of functions that may be used for $\mathcal{F}_g$.

Observation 3 *$\mathcal{F}_g$ may be defined as any function $(x, y) \mapsto \|x - y\|$ for some norm $\|.\|$ over the vector space $\mathbb{R}^d$ such that $\mathcal{F}_g$ satisfies Eqs. 7 - 8.*

The L_p norm satisfies Obs. 3. Hence, we define $\mathcal{F}_g$ as:

$$\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) = \left\|\mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right\|_p \tag{9}$$

In our implementation we use the L_2 norm. Finally, the parameters of the entire model are learned by minimizing the mean squared error (here $\mathbb{T}$ is the training set).

$$\mathscr{L} = \frac{1}{|\mathbb{T}|} \sum_{\forall \langle \mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T} \rangle \in \mathbb{T}} \left(\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) - \textsc{Ged}\left(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}\right)\right)^2 \tag{10}$$

Intuition: Regardless of the graph representations generated by our model, $\mathcal{F}_g$ ensures that the predicted distance is a metric. One the other hand, by training the model to produce embeddings $\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}$ and $\mathbf{Z}_{\mathcal{G}_\mathcal{T}}$ such that $\mathcal{F}_g\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) \approx \textsc{Ged}\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right)$, we enforce a rich structure on the embedding space such that $\mathcal{F}_g$ is also accurate. Thus, $\mathcal{F}_g$ injects an inductive bias satisfying the dual needs of accuracy and preservation of original space properties.

3.1.2 SED

SED satisfies non-negativity and triangle inequality. Following a similar reasoning as above, we define $\mathcal{F}_s$ as follows:

$$\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) = \|ReLU(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}})\|_2 = \left\|\max\left\{0, \mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right\}\right\|_2 \tag{11}$$

Intuitively, for those co-ordinates where the value of $\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}$ is greater than $\mathbf{Z}_{\mathcal{G}_\mathcal{T}}$, a distance penalty is accounted by $\mathcal{F}_s$ in terms of how much those values differ; otherwise $\mathcal{F}_s$ considers 0. This follows the intuition that the SED accounts for those features of $\mathcal{G}_\mathcal{Q}$ that are not in $\mathcal{G}_\mathcal{T}$. Moreover, consistent with SED, the additional features in $\mathcal{G}_\mathcal{T}$ that are not in $\mathcal{G}_\mathcal{Q}$, do not incur any cost.

Lemma 1 *The following properties hold on predicted* SED.

1. $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) \geq 0$
2. $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) = 0 \iff \mathbf{Z}_{\mathcal{G}_\mathcal{Q}} \leq \mathbf{Z}_{\mathcal{G}_\mathcal{T}}$
3. $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) \leq \mathcal{F}(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}) + \mathcal{F}(\mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}})$

PROOF. Properties (1) and (2) follow from the definition of $\mathcal{F}$ itself. Property (3) follows from the fact that we take the L_2 norm. Formally, we state it as follows.

Lemma 2 $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right) \leq \mathcal{F}(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}) + \mathcal{F}(\mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}})$, *where* $\mathcal{F}(\mathbf{x}, \mathbf{y}) = \|ReLU(\mathbf{x} - \mathbf{y})\|$ *for any monotonic norm* $\|.\|$.

PROOF. See App. B.4.

3.2 Characterization of GREED

Importance of pair-independence and siamese architecture: If we do not use a siamese architecture, then the embedding model for the query and the target graphs would be different. Hence, the predicted distance would violate symmetry, and therefore, would not be a metric. Furthermore, if the distance computations are pair-dependent [27, 42, 2], i.e., it jointly learns the embedding of the query and the target, then a single graph may correspond to multiple representations. Hence, it would not be a metric or satisfy the triangle inequality.

Complexity Analysis: The complexity of GED and SED inference in GREED is *linear* to the number of nodes and edges in the query and target graphs (See App. C for derivation). This computation cost is drastically lower than the factorial computation cost of optimal GED and SED. With respect to neural methods for graph similarity [2, 3, 27, 37, 42], all have at least quadratic computation cost, i.e., $O(|\mathcal{V}|^2)$.

Indexing Embeddings: Since the generated embeddings for both GED and SED satisfy triangle inequality, they are indexable leading to fast querying times. We develop an index structure to exploit this property. Due to space limitations, the details are included in App. D. In addition, we also design a *neighborhood decomposition* scheme, which enables fast pruning of the exponential search space § D.4. In § 4.3, we empirically analyze the impact of index structures on querying time.

4 Empirical Evaluation

In this section, we establish the following:

- **Efficacy:** GREED is more accurate than the state of the art approaches for both GED and SED.
- **Efficiency:** GREED is orders of magnitude faster than existing approaches and scales well to graphs with millions of nodes.
- **Scalability:** Pair-independence and indexability further enhances the scalability of GREED and enables it to be run on CPU-only platforms.

Our code base and datasets are available at https://anonymous.4open.science/r/greed/.

4.1 Experimental Setup

We use a machine with an Intel Xeon Gold 6142 processor and GeForce GTX 1080 Ti GPU for all our experiments.

Datasets: Table A lists the datasets used for benchmarking. Further details on the dataset semantics are provided in the App. E. We include a mixture of both *graph databases* (#graphs >1), as well as *single large* graphs (#graphs = 1). `Linux` and `IMDB` are unlabeled. We note that this is the first study to evaluate neural graph distance approaches on million-scale graphs.

Baselines: To evaluate performance in **GED**, we compare with GENN-A* [37], H^2MN[42] and GOTSIM [13]. These are the most recent neural frameworks and have shown better efficacy than other neural approaches such as SIMGNN [2], GRAPHSIM [3], and GMN [27].

For **SED**, no neural approaches exist. However, H^2MN and SIMGNN can be trained by replacing GED with SED along with minor modifications in training. NSC [29] is a method for counting subgraphs using graph embeddings. Since this is a related operation, we use NSC as a baseline by changing the loss function to minimize the RMSE between true and predicted SED. We also use NEUROMATCH [32] as a baseline, which was originally designed to detect subgraph isomorphism. While NEUROMATCH cannot predict SED, it generates a *violation score*, which can be interpreted as the likelihood of the query being subgraph isomorphic to the target. The violation score can be used as a proxy for SED and used in ranking of k-NN (k-NearestNeighbour) queries. Thus, NEUROMATCH comparisons are limited to k-NN queries on SED. GMN [27], GRAPHSIM [3], GOTSIM [13] and GENN-A* are not included since they cannot be easily adapted for SED. See App. F for details.

In the **non-neural** category, we use *mixed integer programming* based method MIP-F2 [25] with a time bound of 0.1 seconds per pair for both GED and SED. MIP-F2 provides the optimal solution given infinite time. We also compare with BRANCH [4], which achieves an excellent trade-off between accuracy and time [5]. BRANCH uses *linear sum assignment problem with error-correction* (LSAPE) to process the search space. We use GEDLIB's [6] implementation of these methods.

Training (and Test) Data Generation: For GED, we use $\langle query, target \rangle$ graph pairs from `IMDB`, `AIDS'`, and `Linux`. Our setup is identical to SIMGNN [2] and H^2MN [42].

For SED, the target graphs are taken from datasets listed in Table A. For the query graph, in `AIDS`, we use known *functional groups* [31]. In the rest of the graph datasets, queries are sampled by performing a random BFS traversal (depth up to 5). Table A shows the average query sizes ($|\mathcal{V}_\mathcal{Q}|$, $|\mathcal{E}_\mathcal{Q}|$).We use *mixed integer programming* method F2 [25] implemented in GEDLIB [6] with a large time limit to generate ground-truth data.

Train-Validation-Test: We use $100K$ query-target pairs for training and $10K$ pairs each for validation and test. All models are trained till validation loss is minimized or there is less than 0.05% change in validation loss over a number of extended epochs. The codebase of all baselines have been obtained from the respective authors. For all baselines, we use the default parameters suggested by the authors. We also conducted experiments on the datasets to ensure that the default parameters indeed provide the best performance. For GREED, we set the number of layers in GIN to 8. The hidden layer dimension is set to 64.

4.2 Prediction Accuracy of SED and GED

Tables 1a and 1b present the accuracy of all techniques on GED and SED in terms of *Root Mean Square Error (RMSE)*. GREED outperforms all other techniques in 9 out of 10 settings across GED and SED. While H^2MN and NSC are the second best performers in SED, GENN-A* performs well in GED. GENN-A*, however, is extremely slow and does not scale on graphs of size beyond 10. H^2MN, thus, provides the second best balance between efficacy and efficiency after GREED. The gap in

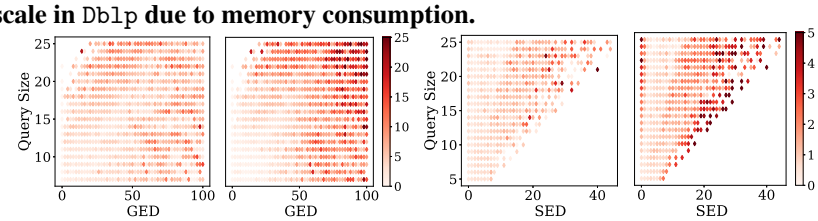
Methods	AIDS'	Linux	IMDB
GREED	**0.796**	0.415	**6.734**
H^2MN	0.994	0.734	86.077
GENN-A*	0.907	**0.267**	NA
GOTSIM	0.996	0.574	37.831
SIMGNN	1.037	0.666	66.250
Branch	3.322	2.474	6.875
MIP-F2	2.929	1.245	82.124

(a) Prediction of GED

Methods	Dblp	Amazon	PubMed	CiteSeer	Cora_ML	Protein	AIDS
GREED	**0.964**	**0.495**	**0.728**	**0.519**	**0.635**	**0.524**	**0.512**
H^2MN	1.470	1.294	1.213	1.502	1.446	0.941	0.749
NSC	NA	2.141	1.095	1.66	1.661	0.662	0.562
SIMGNN	1.482	2.810	1.322	1.781	1.289	1.223	0.696
Branch	2.917	4.513	2.613	3.161	3.102	2.391	1.379
MIP-F2	3.427	5.595	3.399	4.474	3.871	2.249	1.537

(b) Prediction of SED.

Table 1: **(a) Datasets. (b-c) RMSE scores (lower is better) in (a) GED and (b) SED. GENN-A* does not scale on graphs beyond 10 nodes and hence results in IMDB are not reported. NSC does not scale in Dblp due to memory consumption.**



(a) GREED, IMDB **(b)** H^2MN, IMDB **(c)** GREED, Dblp **(d)** H^2MN, Dblp

Figure 3: **Heat Map of RMSE in (a-b) GED and (c-d) SED against query size in IMDB and Dblp.**

accuracy is the highest in IMDB for GED, where GREED is more than 10 times better than the neural baselines. IMDB graphs are significantly denser and larger than AIDS' or Linux. Thus, computing the optimal GED is harder. While all techniques have higher errors in IMDB, the deterioration is more severe in the baselines indicating that GREED scales better with graph sizes.

Impact of Query Size: We next investigate how the accuracy varies against the query size. Intuitively, the task gets harder with query size since the combinatorial space of possible maps increases exponentially. For this analysis, we compare GREED with H^2MN in IMDB and Dblp for GED and SED respectively. GENN-A* fails to scale on both datasets.

In Fig. 3, we plot the *heat map* of RMSE against query graph size. In this plot, each dot corresponds to a query graph $\mathcal{G}_\mathcal{Q}$. The co-ordinate of a query is $(\text{GED}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}), |\mathcal{V}_\mathcal{Q}|)$ (analogously defined for SED). The color of a dot represents the RMSE; the darker the color, the higher is the RMSE. When we compare the heat maps of GREED with H^2MN, we observe that H^2MN is noticeably darker. Furthermore, the concentration of dark colors is noticeably higher on the upper-right corner indicating deterioration with larger query sizes and higher distance values. This indicates that GREED scales better with query sizes and distances.

Visualization: A case study to visually illustrate the efficacy of GREED is provided in App. I.

Range and k-NN queries: To quantify performance in range and k-NN queries, we measure *F1-score (Range query)* and *Kendalls's tau (k-NN)* [22] of the predicted answer set, when compared against the ground truth. In Figs. 4a-4h, we measure the performance in range queries. In SED, GREED consistently outperforms all baselines in F1-score. In GED, the trend remains similar. Although, GENN-A* outperforms GREED for a brief region in Linux, overall, GREED has the highest F1-score. We also note the GENN-A* is not included in Fig. 4h since it fails to scale on IMDB. In k-NN queries (Tables 2a and 2b), GREED outperforms all algorithms in SED. In GED, similar to the trend in range queries, GREED is the dominant method and GENN-A* marginally outperforms GREED in Linux.

Methods	AIDS'	Linux	IMDB
GREED	**0.80**	0.89	**0.87**
H^2MN	0.74	0.88	0.80
GENN-A*	0.75	**0.90**	NA
SIMGNN	0.72	0.86	0.67

(a) Ranking in GED.

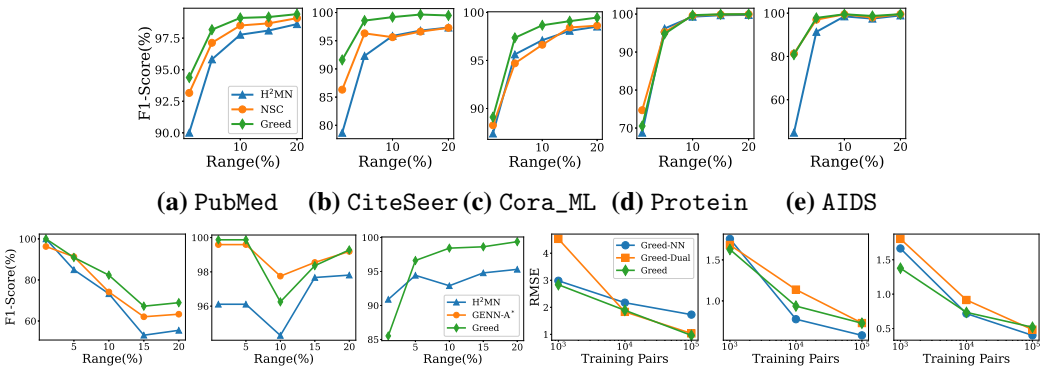
Methods	PubMed	CiteSeer	Cora_ML	Protein	AIDS
GREED	**0.90**	**0.90**	**0.91**	**0.75**	**0.80**
H^2MN	0.87	0.88	0.88	0.70	0.72
NSC	0.89	0.88	0.88	0.74	0.78
SIMGNN	0.85	0.87	0.86	0.63	0.73
NEUROMATCH	0.70	0.75	0.73	0.57	0.59

(b) Ranking in SED.

Table 2: **Kendall's tau scores (higher is better).**

4.3 Efficiency

Tables 3a-3b present the inference times per $10K$ query-target pairs. In this experiment, we do not index embeddings by GREED so that the comparison unearths the raw difference in computation



(a) PubMed **(b)** CiteSeer **(c)** Cora_ML **(d)** Protein **(e)** AIDS

(f) AIDS' **(g)** Linux **(h)** IMDB **(i)** Dblp **(j)** PubMed **(k)** CiteSeer

Figure 4: **F1-score in range queries on SED (a-e) and GED (f-h). The range threshold is set as a percentage of the max distance observed in the test set. The legend for Figs. (a)-(e) is provided in (a) and for (f-h) is provided in (h). (i-k) Ablation study to analyze the impact of siamese architecture and function $\mathcal{F}$. The legend for Figs. (i)-(k) is provided in (i).**

efficiency of solely the neural architectures. As visible, GREED is up to 1800 times faster than the non-neural baselines and up to 10 to 20 times faster than H^2MN, the current state of the art in GED prediction. Also note that GENN-A* is exorbitantly slow (Table 3a). GENN-A* is slower since it not only predicts the GED but also the alignment via an A* search. While the alignment information is indeed useful, computing this information across all graphs in the database may generate redundant information since an user is typically interested only on a small minority of graphs that are in the answer set. In App. H, we discuss this issue in detail.

Methods	AIDS'	Linux	IMDB
GREED	**0.49**	**0.70**	**0.63**
H^2MN	9.50	8.74	8.83
GENN-A*	12190	1340	NA
BRANCH	10.70	8.24	127.90
MIP-F2	593.34	191.88	1173.548

(a) GED

Methods	Dblp	Amazon	PubMed	CiteSeer	Cora_ML	Protein	AIDS
GREED	**6.84**	**1.46**	**1.30**	**1.28**	**1.25**	**0.86**	**0.84**
H^2MN	44.68	23.2	25.79	27.54	29.04	19.33	9.63
NSC	NA	21	35.05	24.46	70.59	21	4
SIMGNN	109.56	47.68	39.80	39.40	40.73	39.02	43.83
BRANCH	626.489	79.25	99.11	155.09	132.98	52.26	12.93
MIP-F2	1979.185	861.95	606.01	827.65	790.01	881.77	360.12

(b) SED

Datasets	Range ($\theta = 2$)				10-NN			
	CPU		GPU		CPU		GPU	
	L-Scan	Indexed	L-Scan	H2MN	L-Scan	Indexed	L-Scan	H2MN
PubMed	0.693	0.56	0.004	26.6	1.01	0.49	0.004	27.5
Amazon	9.09	5.07	0.025	371	11.3	4.75	0.027	372
Dblp	48	20.9	0.070	696	50.4	18.6	0.126	698

(c) Scalability

Table 3: **(a-b) Running times of all methods in seconds per 10k query-target pair. (c) Querying time (s) for SED in the three largest datasets. L-Scan indicates time taken by linear scan in GREED (times differ based on whether executed on CPU or GPU).**

4.4 Pair-independence and Indexability

Here, we showcase how pair-independent embeddings, and ensuring triangle inequality leads to further boost in scalability. For this experiment, we use the three largest datasets of PubMed, Amazon and Dblp. For each dataset, we pre-compute GREED embeddings of all database graphs by exploiting pair-independent embeddings. Such pre-computation is not possible in the neural or non-neural baselines. Furthermore, since the predictions of GREED satisfy triangle inequality, we index the pre-computed embeddings of the database graphs as discussed in § D.1. Consequently, for GREED, we only need to embed the query graph and evaluate $\mathcal{F}$ to make predictions at query time. Table 3c presents the results on range and 10-NN queries. When computations are done on a GPU, GREED is more than 1000 times faster than H^2MN. In the absence of a GPU, H^2MN is practically infeasible since expensive pair-dependent computations are done at query time. In contrast, even on a CPU, through indexing, GREED is ≈ 50 times faster than GPU-based H^2MN. Note that indexing enables up to 3-times speed-up on GREED over linear scan, which demonstrates the gain from ensuring triangle inequality. These results establish that GREED breaks new ground in scalability of neural graph distance computations; not only is it faster, it overcomes the barrier of GPU-dependence and hence better suited for low-resource environments.

4.5 Ablation Study

In this study, we explore the impact of our inductive biases in learning from low-volume data. We create two variants of GREED: **(1)** GREED-Dual trains the two parallel GNN models separately without weight-sharing, and **(2)** GREED-NN uses an MLP instead of $\mathcal{F}$. Both have strictly better representational capacity than GREED, so are expected to match the performance with infinite data. Figs. 4i-4k present the results on SED. The results of the same experiment on GED is provided in Figs. F in the appendix. The RMSE of GREED is consistently better than GREED-Dual, with the difference being more significant at low volumes. This indicates that siamese structure helps. Compared to GREED, GREED-NN achieves marginally better performance at larger train sizes in `PubMed` and `CiteSeer`. However, in `Dblp`, GREED is consistently better. The number of subgraphs in a dataset grows exponentially with the node set size. Hence, an MLP needs growing training data to accurately model the intricacies of this search space. In `Dblp`, even 100k pairs is not enough to improve upon $\mathcal{F}$. Furthermore, since computing GED and SED is NP-hard, generating large volumes of training data is not desirable. Overall, these trends indicate that $\mathcal{F}$ enables better generalization and scalability with respect to accuracy. Furthermore, given that its performance is close to an MLP even on high-volume training data, and it enables indexing, the benefits outweigh the marginal reduction in accuracy. More ablations studies justifying our choice of GIN and the sum-pool layer are provided in App. G.

Sampler	`PubMed`	`CiteSeer`	`Amazon`
BFS	0.728	0.519	0.495
RW	0.508	0.770	0.490
RWR	0.545	0.754	0.299
SHADOW	0.966	0.753	0.830

(a) Query distributions

Method	`PubMed`		`CiteSeer`		`Amazon`	
	$\mathcal{V}_Q \in [0, 50]$	$\mathcal{V}_Q \in [25, 50]$	$\mathcal{V}_Q \in [0, 50]$	$\mathcal{V}_Q \in [25, 50]$	$\mathcal{V}_Q \in [0, 50]$	$\mathcal{V}_Q \in [25, 50]$
GREED-50	1.294	1.917	0.728	0.948	0.638	0.782
GREED-25	2.824	4.999	4.740	9.052	1.152	1.724
H^{2}MN-50	3.1334	5.112	4.9380	8.583	6.014	9.550
H^{2}MN-25	7.417	13.366	10.459	19.787	5.72	9.462
MIP-F2	3.507	6.278	4.831	8.505	6.454	10.293

(b) Query size

Table 4: **(a) RMSE on unseen query distributions. BFS (seen) is the baseline to compare against. (b) RMSE against query sizes. GREED-50 indicates GREED trained on a dataset containing queries of size up to 50. GREED-25 is defined analogously.**

4.6 Generalization to Unseen Query Distributions in SED

We train the model by sampling queries from the graph database through BFS enumerations. *How does* GREED *generalize to unseen distributions?* Towards that end, we generate queries from the three unseen distributions of **(1)** Random Walks (RW), **(2)** Random Walks with Restarts (RWR), and **(3)** SHADOW [40] (See App. J for details on the sampling strategies). We first note that in `AIDS`, we use real queries of functional groups, and thus the good performance in `AIDS` indicates good generalizability. In Table 4a, we more exhaustively analyze this aspect. As visible, the errors remain low. Even more surprisingly, the errors on RW and RWR are better than the train distribution of BFS itself. This indicates good generalization to unseen distributions.

4.7 Generalizability to Unseen, Larger Query Sizes:

Generating training data for learning GED and SED is expensive since optimal distance computations are NP-hard. Hence, a desirable property would be to learn from small graphs and then generalize to larger unseen graphs. We evaluate this ability for GREED and H^{2}MN. Table 4b provides the numbers. We notice that although there is some deterioration in the quality for query sizes in the range $[25, 50]$ when compared to the entire set, it is not severe (GREED-50 in Table 4b). However, if the train set only contains queries till size 25 and we deploy the learned model to infer on queries of larger unseen sizes, the drop in quality is significant (GREED-25 in Table 4b). This drop is even more dramatic in H^{2}MN. On the positive side, GREED remains superior to the optimal non-neural approach (MIP-F2) when run with a generous time limit of 60 seconds per query. Overall, this experiment highlights one direction that needs further study and improvement.

5 Conclusions, Limitation, and Future Directions

The problem of learning graph distances from their embeddings has seen much interest over the last few years. This thread of research is important since it allows us to overcome the bottleneck of exponential graph alignment space. Our experiments clearly establish GREED as the state of the art for both GED and SED (See App. K for preliminary results on *maximum common subgraph similarity*). In addition, it is significantly faster and provides better theoretical correspondence between properties of the original space and predicted space. One clear direction of future work that emerges from our experiments is that GREED, and existing methods of graph distance learning, do not generalize well to unseen larger query sizes. We hope to address this limitation next.

References

[1] Jiyang Bai and Peixiang Zhao. Tagsim: Type-aware graph similarity learning and computation. *Proc. VLDB Endow.*, 15(2):335–347, 2021.

[2] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *wSDM*, WSDM '19, page 384–392, 2019.

[3] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. *AAAI*, pages 3219–3226, Apr. 2020.

[4] David B Blumenthal. New techniques for graph edit distance computation. *arXiv preprint arXiv:1908.00265*, 2019.

[5] David B Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29(1):419–458, 2020.

[6] David B Blumenthal, Sébastien Bougleux, Johann Gamper, and Luc Brun. Gedlib: a c++ library for graph edit distance computation. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 14–24. Springer, 2019.

[7] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[8] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large sparql query logs. *Proc. VLDB Endow.*, 11(2):149–161, 2017.

[9] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017. Advances in Graph-based Pattern Recognition.

[10] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6:737–744, 1993.

[11] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[12] Évariste Daller, Sébastien Bougleux, Benoit Gaüzère, and Luc Brun. Approximate Graph Edit Distance by Several Local Searches in Parallel. In *7th International Conference on Pattern Recognition Applications and Methods*, 2018.

[13] Khoa D. Doan, Saurav Manchanda, Suchismit Mahapatra, and Chandan K. Reddy. Interpretable graph similarity computation via differentiable optimal alignment of node embeddings. In *SIGIR*, page 665–674, 2021.

[14] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. D-index: Distance searching index for metric data sets. *Multim. Tools Appl.*, 21(1):9–33, 2003.

[15] Chi Thang Duong, Trung Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. Efficient streaming subgraph isomorphism with graph neural networks. 14(5):730–742, January 2021.

[16] Christopher L. Ebsch, Joseph A. Cottam, Natalie C. Heller, Rahul D. Deshmukh, and George Chin. Using graph edit distance for noisy subgraph matching of semantic property graphs. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2520–2525, 2020.

[17] Charles Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, page 147–153, 2003.

[18] Ali S. Hadi. Finding groups in data: An introduction to cluster analysis. *Technometrics*, 34:111–112, 1991.

[19] Huahai He and Ambuj K Singh. Closure-tree: An index structure for graph queries. In *ICDE*, pages 38–38. IEEE, 2006.

[20] Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[21] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[22] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[23] Wojciech Kwedlo and Pawel J. Czochanski. A hybrid mpi/openmp parallelization of k -means algorithms accelerated using the triangle inequality. *IEEE Access*, 7:42280–42297, 2019.

[24] AA Leman and B Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[25] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017.

[26] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[27] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, pages 3835–3845, 2019.

[28] Yongjiang Liang and Peixiang Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*, pages 783–794, 2017.

[29] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *KDD*, page 1959–1969, 2020.

[30] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML workshop on Graph Representation Learning and Beyond*, 2020.

[31] Sayan Ranu and Ambuj K Singh. Mining statistically significant molecular substructures for efficient molecular classification. *Journal of chemical information and modeling*, 49(11):2537–2550, 2009.

[32] Rex, Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching, 2020.

[33] Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. Interpretable neural subgraph matching for graph retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8115–8123, 2022.

[34] Aravind Sankar, Sayan Ranu, and Karthik Raman. Predicting novel metabolic pathways through subgraph mining. *Bioinformatics*, 33(24):3955–3963, 2017.

[35] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40:175–179, 1991.

[36] Lichen Wang, Bo Zong, Qianqian Ma, Wei Cheng, Jingchao Ni, Wenchao Yu, Yanchi Liu, Dongjin Song, Haifeng Chen, and Yun Fu. Inductive and unsupervised representation learning on graph structured objects. In *ICLR*, 2020.

[37] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *CVPR*, pages 5241–5250, 2021.

[38] Haibo Xiu, Xiao Yan, Xiaoqiang Wang, James Cheng, and Lei Cao. Hierarchical graph matching network for graph similarity computation, 2020.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[40] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor K. Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. In *NeurIPS*, 2021.

[41] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2(1):25–36, 2009.

[42] Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. In *KDD*, page 2274–2284, 2021.

[43] Xiang Zhao, Chuan Xiao, Xuemin Lin, Qing Liu, and Wenjie Zhang. A partition-based approach to structure similarity search. *VLDB*, 7(3):169–180, 2013.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] For additional proofs, check Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For additional details please check Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please check Sec.4.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] MIT License
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Link to codebase in § 4
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A GED and SED

The computation of GED relies on a *graph mapping*.

Definition 1 (Graph Mapping) *Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, let $\tilde{\mathcal{G}}_1 = (\tilde{\mathcal{V}}_1, \tilde{\mathcal{E}}_1, \tilde{\mathcal{L}}_1)$ and $\tilde{\mathcal{G}}_2 = (\tilde{\mathcal{V}}_2, \tilde{\mathcal{E}}_2, \tilde{\mathcal{L}}_2)$ be obtained by adding dummy nodes and edges (labeled with ϵ) to $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, such that $|\mathcal{V}_1| = |\mathcal{V}_2|$ and $|\mathcal{E}_1| = |\mathcal{E}_2|$. A node mapping between $\mathcal{G}_1$ and $\mathcal{G}_2$ is a bijection $\pi : \tilde{\mathcal{G}}_1 \rightarrow \tilde{\mathcal{G}}_2$ where **(i)** $\forall v \in \tilde{\mathcal{V}}_1, \pi(v) \in \tilde{\mathcal{V}}_2$ and at least one of v and $\pi(v)$ is not a dummy; **(ii)** $\forall e = (v_1, v_2) \in \tilde{\mathcal{E}}_1, \pi(e) = (\pi(v_1), (\pi(v_2))) \in \tilde{\mathcal{E}}_2$ and at least one of e and $\pi(e)$ is not a dummy.*

Example 1 *Fig. 1 shows a graph mapping. Edge mappings can be trivially inferred.*

Definition 2 (Graph Edit Distance (GED) under mapping π) GED *between $\mathcal{G}_1$ and $\mathcal{G}_2$ under π is*

$$\text{GED}_\pi(\mathcal{G}_1, \mathcal{G}_2) = \sum_{v \in \tilde{V}_1} d(\mathcal{L}(v), \mathcal{L}(\pi(v))) + \sum_{e \in \tilde{E}_1} d(\mathcal{L}(e), \mathcal{L}(\pi(e))) \tag{12}$$

where $d : \Sigma \times \Sigma \rightarrow \mathbb{R}_0^+$ is a distance function over the label set. $d(\ell_1, \ell_2)$ models an insertion if $\ell_1 = \epsilon$, deletion if $\ell_2 = \epsilon$ and replacement if $\ell_1 \neq \ell_2$ and neither ℓ_1 nor ℓ_2 is a dummy.

We assume d to be a binary function, where $d(\ell_1, \ell_2) = 1$ if $\ell_1 \neq \ell_2$, otherwise, 0.

Definition 3 (Graph Edit Distance (GED)) GED *is the minimum distance under all mappings.*

$$\text{GED}(\mathcal{G}_1, \mathcal{G}_2) = \min_{\forall \pi \in \Phi(\mathcal{G}_1, \mathcal{G}_2)} \text{GED}_\pi(\mathcal{G}_1, \mathcal{G}_2) \tag{13}$$

$\Phi(\mathcal{G}_1, \mathcal{G}_2)$ denotes the set of all possible node maps from $\mathcal{G}_1$ to $\mathcal{G}_2$.

Definition 4 (Subgraph Edit Distance (SED)) SED *is the minimum* GED *over all subgraphs of $\mathcal{G}_2$.*

$$\text{SED}(\mathcal{G}_1, \mathcal{G}_2) = \min_{\mathcal{S} \subseteq \mathcal{G}_2} \text{GED}(\mathcal{G}_1, \mathcal{S}) \tag{14}$$

Observation 4 ***(i)*** $\text{GED}(\mathcal{G}_1, \mathcal{G}_2) \geq 0$, ***(ii)*** $\text{SED}(\mathcal{G}_1, \mathcal{G}_2) \geq 0$.

Observation 5 ***(i)*** $\text{GED}(\mathcal{G}_1, \mathcal{G}_2) = 0$ *iff $\mathcal{G}_1$ is isomorphic to $\mathcal{G}_2$,* ***(ii)*** $\text{SED}(\mathcal{G}_1, \mathcal{G}_2) = 0$ *iff $\mathcal{G}_1$ is subgraph isomorphic to $\mathcal{G}_2$.*

B Additional Proofs

B.1 Proof of Theorem. 1

Our proof relies on two lemmas.

PROOF of Theorem 1. It suffices to prove **(i)** $\text{SED}(\mathcal{G}_1, \mathcal{G}_2) \geq \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$ and **(ii)** $\widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2) \geq \text{SED}(\mathcal{G}_1, \mathcal{G}_2)$.

(i) Let $\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S}, \mathcal{L}_\mathcal{S}) \subseteq \mathcal{G}_2$ be the subgraph minimizing $\text{SED}(\mathcal{G}_1, \mathcal{S})$ (Recall Eq. 14). Consider the mapping π from $\mathcal{G}_1$ to $\mathcal{S}$ corresponding to $\text{SED}(\mathcal{G}_1, \mathcal{S})$ (and hence $\text{GED}(\mathcal{G}_1, \mathcal{S})$ as well). We extend π to define a mapping $\widehat{\pi}$ from $\mathcal{G}_1$ to $\mathcal{G}_2$ by mapping of all nodes in set $\mathcal{V}_2 \setminus \mathcal{V}_\mathcal{S}$ to dummy nodes in $\mathcal{G}_1$; the edge mappings are defined analogously.

Under this construction, $\text{SED}(\mathcal{G}_1, \mathcal{S}) = \text{GED}(\mathcal{G}_1, \mathcal{G}_2) = \widehat{\text{GED}}_{\widehat{\pi}}(\mathcal{G}_1, \mathcal{G}_2) \geq \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$. This follows from the property that under $\widehat{d}$, insertion costs are zero, that is $\widehat{d}(\epsilon, \ell) = 0$. Thus, the additional mappings introduced in $\widehat{\pi}$ do not incur additional costs under $\widehat{d}$.

(ii) Consider $\mathcal{S} \subseteq \mathcal{G}_2$ and a mapping π from $\mathcal{G}_1$ to $\mathcal{S}$ such that $\text{GED}_\pi(\mathcal{G}_1, \mathcal{S}) = \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$. The existence of such a subgraph is guaranteed (See Lemma 3). From the definition of GED, $\text{GED}_\pi(\mathcal{G}_1, \mathcal{S}) \geq \text{GED}(\mathcal{G}_1, \mathcal{S})$. Furthermore, since $\mathcal{S} \subseteq \mathcal{G}_2$, $\text{GED}(\mathcal{G}_1, S) \geq \text{SED}(\mathcal{G}_1, \mathcal{G}_2)$. Combining all these results, we have $\widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2) \geq \text{GED}(\mathcal{G}_1, S) \geq \text{SED}(\mathcal{G}_1, \mathcal{G}_2)$.
Hence, the claim is proved. □

B.2 Proof of Thm 2.

PROOF. From Thm. 1, we know $\text{SED}(\mathcal{G}_1, \mathcal{G}_2) = \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$. Combining Obs. 1 with Theorem 1, if we show $\widehat{d}(\ell_1, \ell_3) \leq \widehat{d}(\ell_1, \ell_2) + \widehat{d}(\ell_2, \ell_3)$, then the triangle inequality of SED is established. We divide the proof into four cases:

(i) None of ℓ_1, ℓ_2, ℓ_3 is ϵ. Hence, $\widehat{d}(\ell_1, \ell_3) = d(\ell_1, \ell_3)$ and the triangle inequality is satisfied.
(ii) $\ell_1 = \epsilon$. The LHS is 0 and hence the triangle inequality is satisfied.
(iii) $\ell_1 \neq \epsilon$ and $\ell_2 = \epsilon$. LHS≤ 1 and RHS$= 1$. Hence, satisfied.
(iv) Only $\ell_3 = \epsilon$. Here, LHS$= 1$ and RHS≥ 1.
These four cases cover all possible situations and hence, the triangle inequality is established. □

B.3 Proof of Lemma 3

Lemma 3 *There exists a subgraph $\mathcal{S}$ of $\mathcal{G}_2$ and a node map π from $\mathcal{G}_1$ to $\mathcal{S}$ such that* $\text{GED}_\pi(\mathcal{G}_1, \mathcal{S}) = \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$.

PROOF. Let π' be a node map from $\mathcal{G}_1$ to $\mathcal{G}_2$ corresponding to $\widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$. Let $h_1, \cdots, h_l$ be the nodes of $\mathcal{G}_2$ which are inserted in π'. Construct subgraph $\mathcal{S}$ of $\mathcal{G}_2$ by removing nodes $h_1, \cdots, h_l$ and their incident edges from $\mathcal{G}_2$. Let π be the node map from $\mathcal{G}_1$ to $\mathcal{S}$ which is obtained by removing $h'_1, \cdots, h'_l$ from the domain and $h_1, \cdots, h_l$ from the co-domain of π. Since insertion costs are 0 in $\widehat{d}$ and π contains only non-insert operations, then $\text{GED}_\pi(\mathcal{G}_1, \mathcal{S}) = \widehat{\text{GED}}(\mathcal{G}_1, \mathcal{G}_2)$. Hence, the claim is proved. □

B.4 Proof of Lemma 2.

PROOF. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ be vectors of dimension n. We use the notation $\mathbf{x}[i]$ to denote the ith coordinate of $\mathbf{x}$. We observe that :

$$\begin{aligned}(\text{ReLU}(\mathbf{x}) + \text{ReLU}(\mathbf{y}))[i] &= \text{ReLU}(\mathbf{x})[i] + \text{ReLU}(\mathbf{y})[i] = \text{ReLU}(\mathbf{x}[i]) + \text{ReLU}(\mathbf{y}[i]) \\ &\geq \text{ReLU}(\mathbf{x}[i] + \mathbf{y}[i]) = (\text{ReLU}(\mathbf{x} + \mathbf{y}))[i]\end{aligned}$$

Since $\|.\|$ is monotonic, this implies $\|\text{ReLU}(\mathbf{x}) + \text{ReLU}(\mathbf{y})\| \geq \|\text{ReLU}(\mathbf{x} + \mathbf{y})\|$. Using the triangle inequality for $\|.\|$, we get:

$$\|\text{ReLU}(\mathbf{x})\| + \|\text{ReLU}(\mathbf{y})\| \geq \|\text{ReLU}(\mathbf{x}) + \text{ReLU}(\mathbf{y})\| \geq \|\text{ReLU}(\mathbf{x} + \mathbf{y})\| \tag{15}$$

Substituting $\mathbf{x} = \mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}, \mathbf{y} = \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}}$, we get,

$$\|\text{ReLU}(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}})\| + \|\text{ReLU}(\mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}})\| \geq \|\text{ReLU}(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}} - \mathbf{Z}_{\mathcal{G}_\mathcal{T}})\|.$$

This implies $\mathcal{F}(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}) + \mathcal{F}(\mathbf{Z}_{\mathcal{G}_{\mathcal{T}'}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}) \geq \mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_{\mathcal{G}_\mathcal{T}}\right)$. □

C Complexity Analysis

For this analysis, we make the simplifying assumption that the hidden dimension in the Pre-MLP, GIN and Post-MLP are all d. The average density of the graph is g. The number of hidden layers in Pre-MLP, and Post-MLP are L, and k in GIN.

The computation cost per node for each of these components are as follows.

- **Pre-MLP:** The operations in the MLP involve linear transformation over the input vector $\mathbf{x}_v$ of dimension $|\Sigma|$, followed by non-linearity. This results in $O(|\mathcal{V}|(|\Sigma| \cdot d + d^2 L))$ cost.
- **GIN:** GIN aggregates information from each of the neighbors, which consumes $O(d \cdot g)$ time. The linear transformation consumes an additional $O(d^2)$ time. Applying non-linearity takes $O(d)$ time since it is a linear pass over the hidden dimensions. Finally these operations are repeated over each of the k hidden layers, results in a total $O(k(d^2 + dg))$ computation time per node. Across, all nodes, the total cost is $O(|\mathcal{V}|kd^2 + |\mathcal{E}|kd)$ time. The degree g terms gets absorbed since each edge passes message twice across all nodes.
- **Concatenation:** This step consumes $O(kd)$ time per node.
- **Pool:** Pool iterates over the GIN representation of each node requiring $O(|\mathcal{V}|dk)$ time.
- **Post-MLP:** The final MLP takes dk dimensional vector as input and maps it to a d dimensional vector over L layers. This consumes $O(kd^2 + d^2 L)$ time.

Algorithm 1 BUILDINDEX

Input: Embeddings $\mathbb{D}$ of graphs
Output: Root node of the constructed tree
1: **if** $\mathbb{D} = \emptyset$ **then return** NULL
2: $\mathbf{Z}_{\mathcal{P}} \leftarrow$ arbitrary embedding in $\mathbb{D}$ as pivot
3: $m_1 \leftarrow \text{MEDIAN}(\{\mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}}) : \mathbf{Z}_{\mathcal{G}} \in \mathbb{D} \setminus \{\mathbf{Z}_{\mathcal{P}}\}\})$
4: $m_2 \leftarrow \text{MEDIAN}(\{\mathcal{F}_s(\mathbf{Z}_{\mathcal{G}}, \mathbf{Z}_{\mathcal{P}}) : \mathbf{Z}_{\mathcal{G}} \in \mathbb{D} \setminus \{\mathbf{Z}_{\mathcal{P}}\}\})$
5: $\mathbb{D}_1 \leftarrow \{\mathbf{Z}_{\mathcal{G}} : \mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}}) \leq m_1, \mathcal{F}_s(\mathbf{Z}_{\mathcal{G}}, \mathbf{Z}_{\mathcal{P}}) \leq m_2\}$
6: $\mathbb{D}_2 \leftarrow \{\mathbf{Z}_{\mathcal{G}} : \mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}}) \leq m_1, \mathcal{F}_s(\mathbf{Z}_{\mathcal{G}}, \mathbf{Z}_{\mathcal{P}}) > m_2\}$
7: $\mathbb{D}_3 \leftarrow \{\mathbf{Z}_{\mathcal{G}} : \mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}}) > m_1, \mathcal{F}_s(\mathbf{Z}_{\mathcal{G}}, \mathbf{Z}_{\mathcal{P}}) \leq m_2\}$
8: $\mathbb{D}_4 \leftarrow \{\mathbf{Z}_{\mathcal{G}} : \mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}}) > m_1, \mathcal{F}_s(\mathbf{Z}_{\mathcal{G}}, \mathbf{Z}_{\mathcal{P}}) > m_2\}$
9: **for** $i = 1$ to 4 **do**
10: $\quad t_i \leftarrow \text{BUILDINDEX}(\mathbb{D}_i)$
11: **Return** $\text{NODE}(\mathbf{Z}_{\mathcal{P}}, m_1, m_2, t_1, t_2, t_3, t_4)$

Combining all these factors, the total inference complexity for a graph is $O(|\mathcal{V}|(|\Sigma| \cdot d + d^2L + kd^2) + |\mathcal{E}|kd)$. This operation is repeated on both the query and target graphs to compute their embeddings, on which distance function $\mathcal{F}$ is operated. Thus, the final cost is $O(n(|\Sigma| \cdot d + d^2L + kd^2) + mkd)$, where $n = |\mathcal{V}_{\mathcal{Q}}| + |\mathcal{V}_{\mathcal{T}}|$ and $m = |\mathcal{E}_{\mathcal{Q}}| + |\mathcal{E}_{\mathcal{T}}|$.

D Querying in the Embedding Space

We assume the standard querying setup where the database graphs are known apriori, while the query graph is unseen and provided at query time. Since GREED generates pair-independent embeddings, representations of the database graphs can be generated apriori and stored. Furthermore, due to both the predicted GED and SED satisfying the triangle inequality, the embeddings can be indexed. Thus, at query time, we need to perform only two operations: **(1)** embed the query graph $\mathcal{G}_{\mathcal{Q}}$, and **(2)** scan the database embeddings against the query embedding to compute the answer set. We focus the discussion on the two most common database queries of *range* and *k-NN* queries.

Definition 5 (Range Query) *Given a database $\mathbb{D} = \{\mathbf{Z}_{\mathcal{G}_1}, \cdots, \mathbf{Z}_{\mathcal{G}_n}\}$ of graph embeddings, a query graph $\mathcal{G}_{\mathcal{Q}}$ and a threshold θ, find the answer set $\mathbb{A} = \{\mathcal{G}_i \mid \mathcal{F}_g(\mathbf{Z}_{\mathcal{G}_i}, \mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}) \leq \theta\}$.*

Definition 6 (k-NN Query) *Given a database $\mathbb{D} = \{\mathbf{Z}_{\mathcal{G}_1}, \cdots, \mathbf{Z}_{\mathcal{G}_n}\}$ of graph embeddings, a query graph $\mathcal{G}_{\mathcal{Q}}$ and k, find the database graphs with the k smallest distance to $\mathcal{G}_{\mathcal{Q}}$ as per $\mathcal{F}_g(\mathbf{Z}_{\mathcal{G}_i}, \mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}})$.*

The above definitions can be adopted for SED by using $\mathcal{F}_s$. For the rest of the discussion, we assume $\mathcal{F}_s$ since due to asymmetry, SED requires some additional considerations.

D.1 Indexing

We exploit the triangle inequality of GED and SED to index the database embeddings. Alg. 1 presents the pseudocode. We choose a random embedding $\mathbf{Z}_{\mathcal{P}} \in \mathbb{D}$ as the *pivot* (line 2), based on which we split the remaining embeddings into four groups (lines 3-8). This process continues recursively on each group (lines 9-10) till a partition gets empty (line 1). Note that in GED the distances are

Algorithm 2 RANGEQUERY

Input: Query embedding $\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}$, threshold θ, root node $t = \text{NODE}(\mathbf{Z}_{\mathcal{P}}, m_1, m_2, t_1, t_2, t_3, t_4)$
Output: $\mathbb{A} \leftarrow \{\mathbf{Z}_{\mathcal{G}} \mid \mathcal{F}_s(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{G}}) \leq \theta\}$
if $t = \text{NULL}$ **then return** $\emptyset$
if $\mathcal{F}_s(\mathbf{Z}_{\mathcal{P}}, \mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}) \leq m_1 - \theta$ **then**
$\quad$ mark t_3 and t_4 for pruning
if $\mathcal{F}_s(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{P}}) > m_2 + \theta$ **then**
$\quad$ mark t_1 and t_3 for pruning
if $\mathcal{F}_s(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \mathbf{Z}_{\mathcal{P}}) \leq \theta - m_1$ **then**
$\quad \mathbb{A} \leftarrow \mathbb{A} \cup \mathbb{D}_1 \cup \mathbb{D}_2$
$\quad$ mark t_1 and t_2 for pruning
else
$\quad \mathbb{A} \leftarrow \emptyset$
for $i = 1$ to 4 **do**
$\quad$ **if** t_i is not marked for pruning **then**
$\quad\quad \mathbb{A} \leftarrow \mathbb{A} \cup \text{RANGEQUERY}\left(\mathbf{Z}_{\mathcal{G}_{\mathcal{Q}}}, \theta, t_i\right)$
Return $\mathbb{A}$

Name	$Avg.\lvert\mathcal{V}\rvert$	$Avg.\lvert\mathcal{E}\rvert$	$\lvert\Sigma\rvert$	#Graphs	$Avg.\lvert\mathcal{V}_\mathcal{Q}\rvert$	$Avg.\lvert\mathcal{E}_\mathcal{Q}\rvert$
Dblp	$1.66M$	$7.2M$	8	1	15	14
Amazon	$334k$	$925k$	1	1	12	16
PubMed	$19.7k$	$44.3k$	3	1	12	11
CiteSeer	$4.2k$	$5.3k$	6	1	12	12
Cora_ML	$3k$	$8.2k$	7	1	11	11
Protein	38	70	3	1,071	9	11
AIDS	14	15	38	1,811	7	7
AIDS'	9	9	29	700	9	9
Linux	8	7	1	1,000	8	7
IMDB	13	65	1	1,500	13	65

Table A: Datasets

symmetric and hence m_1 and m_2 will converge. Consequently, we will have two partitions at each node instead of four.

D.2 Range Query

From the triangle inequality, we can infer the lower bounds listed in lines 2 and 4 of Alg. 2. Hence, if these bounds are larger than θ, the corresponding sub-trees are pruned. Similarly, if the upper bound is smaller than θ (line 6), the entire sub-tree is added to the answer set (line 7). Otherwise, we recurse (lines 11-13).

D.3 k-NN query

k-NN utilizes the same bounds from Alg. 2 to prune and prioritize the search space. However, exploration proceeds in a *best-first search* manner. Alg. 3 presents the pseudocode. Alg. 3 maintains two priority-queues; one to keep track of the k-NN till the current stage of the search process ($\mathbb{A}$ in line 1), and the second to store index nodes in ascending order of their lower bound distance ($Cands$ in line 2). We pop the *best* node $\mathcal{P}$ from $Cands$ and include it to the answer set if the distance is within top-k (lines 6-7). Further, the sub-tree at $\mathcal{P}$ is processed if it satisfies the lower bound criteria (lines 8-15). The search ends when either $Cands$ is either empty or the lower bound of the top-most node is larger than the k-th distance in $\mathbb{A}$ (line 4). In case of GED, LB_1 (line 8) and LB_2 (line 9) will converge to the same value due to symmetry .

D.4 Scaling SED to million-scale graphs

SED is typically encountered in situations where the query is a small graph (< 50 nodes) [8, 30] and the target graph is a single large graph, potentially containing millions of nodes and edges. To scale to million-sized target graphs, we perform *neighborhood decomposition*. Specifically, we extract the k-hop neighborhood $\mathcal{G}_v$ around each node $v \in \mathcal{V}_\mathcal{T}$ in the target graph $\mathcal{G}_\mathcal{T}$, embed them into feature space using GREED, and then indexed as outlined § D.1. The distance between query $\mathcal{G}_\mathcal{Q}$ and $\mathcal{G}_\mathcal{T}$ is computed as:

Algorithm 3 k-NN

Input: Query embedding $\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}$, k, root node $t = \text{NODE}(\mathbf{Z}_\mathcal{P}, m_1, m_2, t_1, t_2, t_3, t_4)$
Output: $\mathbb{A} \leftarrow$ top-k closest embeddings in $\mathbb{D}$
1: $\mathbb{A} \leftarrow$ A priority queue of size up to k. Stores entries in descending order of distance.
2: $Cands \leftarrow$ A priority queue. Stores entries in ascending order of distance lower bound.
3: $Cands$.insert$\left(\left\langle t, \mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_\mathcal{P}\right)\right\rangle\right)$
4: **while** $Cands.size() > 0$ **And** $Cands.top().LB < \mathbb{A}.top().distance$ **do**
5: $\quad \mathcal{P} \leftarrow Cands.pop()$
6: $\quad$ **if** $|\mathbb{A}| < k$ **Or** $\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_\mathcal{P}\right) < \mathbb{A}.top().distance$ **then**
7: $\quad\quad \mathbb{A}.insert\left(\left\langle \mathcal{P}, \mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_\mathcal{P}\right)\right\rangle\right)$
8: $\quad LB_1 \leftarrow |\mathcal{F}_s\left(\mathbf{Z}_\mathcal{P}, \mathbf{Z}_{\mathcal{G}_\mathcal{Q}}\right) - m_1|$
9: $\quad LB_2 \leftarrow |\mathcal{F}_s\left(\mathbf{Z}_{\mathcal{G}_\mathcal{Q}}, \mathbf{Z}_\mathcal{P}\right) - m_2|$
10: $\quad LB = \max\{LB_1, LB_2\}$
11: $\quad$ **if** $LB \leq \theta$ **then**
12: $\quad\quad Cands.insert\left(\langle t_1, LB\rangle\right)$
13: $\quad\quad Cands.insert\left(\langle t_2, LB\rangle\right)$
14: $\quad\quad Cands.insert\left(\langle t_3, LB\rangle\right)$
15: $\quad\quad Cands.insert\left(\langle t_4, LB\rangle\right)$
16: **Return** $\mathbb{A}$

$$\mathcal{F}_s\left(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}\right) = \min_{\forall \mathcal{G}_v \in \mathcal{G}_\mathcal{T}} \left\{\mathcal{F}_s\left(\mathcal{G}_\mathcal{Q}, \mathcal{G}_v\right)\right\} \tag{16}$$

We next show that as long as the k-hop neighborhoods are *sufficiently large*, the proposed neighborhood decomposition strategy is optimal.

Theorem 3 *A nearest subgraph $\mathcal{S} \subseteq \mathcal{G}_\mathcal{T}$ to $\mathcal{G}_\mathcal{Q}$ can be found in an $l/2$-hop neighborhood $\mathcal{G}_v \subseteq \mathcal{G}_\mathcal{T}$, centered at some $v \in \mathcal{V}_{\mathcal{G}_\mathcal{T}}$, where l is the length of the longest path in $\mathcal{G}_\mathcal{Q}$.*

PROOF. From Thm. 1, $\text{SED}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}) = \widehat{\text{GED}}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T})$. Let π be the node map produced by $\widehat{\text{GED}}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T})$. Let us now consider the subgraph $\mathcal{S} \subseteq \mathcal{G}_\mathcal{T}$ induced by all node maps of π that are not inserts. It follows that $\text{SED}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}) = \widehat{\text{GED}}(\mathcal{G}_\mathcal{Q}, \mathcal{G}_\mathcal{T}) = \text{SED}(\mathcal{G}_\mathcal{Q}, \mathcal{S})$. Since there are no inserts, the topology of $\mathcal{S}$ is a subgraph of $\mathcal{G}_1$ (i.e., subgraph isomorphic if we ignore label information). Hence, the diameter of $\mathcal{S}$ is $\leq$ the length l of the longest path in $\mathcal{G}_\mathcal{Q}$. Since $\mathcal{S} \subseteq \mathcal{G}_\mathcal{T}$, $\mathcal{S}$ is contained in some $l/2$ neighborhood $\mathcal{G}_v$ of $\mathcal{G}_\mathcal{T}$. □

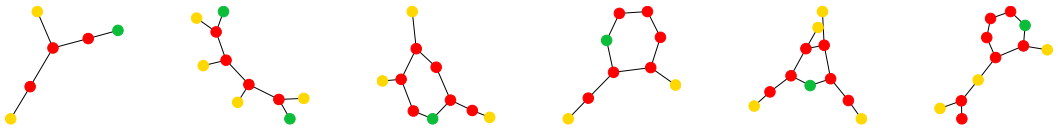
Finding the length of the longest path is NP-hard. However, we do not need to find the exact length of the longest path: any upper bound suffices. Moreover we do not even need the length of the longest path for l. The nearest subgraph having a diameter equal to l is rare. In practice, the diameter of the nearest subgraph is unlikely to be much higher than the diameter of the query itself.

E Datasets

`Dblp`**:** `Dblp` is a co-authorship network where each node is an author and two authors are connected by an edge if they have co-authored a paper. The label of a node is the venue where the authors has published most frequently. The dataset has been obtained from https://www.aminer.org/citation.
`Amazon` **[26]:** Each node in `Amazon` represents a product and two nodes are connected by an edge if they are frequently co-purchased. The graph is unlabeled and hence equivalent to a graph containing a single label on all nodes.
`PubMed`**:** `PubMed` dataset is a citation network which consists of scientific publications from `PubMed` database pertaining to diabetes classified into one of three classes.
`Protein`**:** `Protein` dataset consists of protein graphs. Each node is labeled with a one of three functional roles of the protein.
`AIDS`**:** `AIDS` dataset consists of graphs constructed from the AIDS antiviral screen database. These graphs representing molecular compounds with Hydrogen atoms omitted. Atoms are represented as nodes and chemical bonds as edges.
`AIDS'`**:** `AIDS'` dataset is another collection of graphs constructed from the AIDS antiviral screen database. The graphs and their properties differ from those in `AIDS`. These graphs also represent chemical compound structures.
`CiteSeer`**:** `CiteSeer` is a citation network which consists of scientific publications classified into one of six classes. Generally a smaller version is used for this dataset, but we use the larger version from [7].
`Cora_ML`**:** Cora dataset is a citation dataset consisting of many scientific publications classified into one of seven classes based on paper topic. `Cora_ML` is a smaller datset extracted from Cora [7].
`Linux`**:** `Linux` dataset is a collection of program dependence graphs, where each graph is a function and nodes represent statements while edges represent dependency between statements.
`IMDB`**:** `IMDB` dataset is a collection of ego-networks of actors/actresses that have appeared together in any movie.

F Baselines

GMN explicitly assumes the distance function to be symmetric, which violates SED. GRAPHSIM has an assumption that a large difference in size of the query and target graphs leads to a large distance, which is not true in SED. In GOTSIM, there is an explicit assumption of modeling a symmetric distance function since they use cosine similarity to compare node neighborhoods The normalization factor in Eq. 6 of [13] is also based on whole graph matching. Finally, GENN-A* is not included for SED since it does not scale on graphs beyond 10 nodes (See § 4.3).



(a) Query **(b)** Rank 1 **(c)** Rank 2 **(d)** Rank 3 **(e)** Rank 4 **(f)** Rank 5

Figure E: **Visualizations of query and resulting matches produced by GREED. Red, Green and Yellow colors indicate Carbon, Nitrogen and Oxygen atoms respectively. The actual and predicted SED for the target graphs are (b)** $0, 0.4$**, (c)** $0, 0.5$**, (d)** $1, 0.6$**, (e)** $0, 0.6$ **and (f)** $1, 0.6$**.**

G Ablation Study

Impact of GIN: To highlight the importance of GIN, we conduct ablation studies by replacing the GIN convolution layers in the model with several other convolution layers. As visible in the Table B, GIN consistently achieves the best accuracy. This is not surprising since GIN is provably the most expressive among GNNs in distinguishing graph structures (essential to SED or GED computation) and is as powerful as the Weisfeiler-Lehman Graph Isomorphism test [39].

Methods	CiteSeer (SED)	PubMed (SED)	Amazon (SED)	IMDB (GED)
GREED (GIN)	**0.519**	**0.728**	**0.495**	**6.734**
GREED-GCN	0.556	0.756	0.532	12.151
GREED-GRAPHSAGE	1.364	1.156	1.841	91.312
GREED-GAT	1.294	1.259	1.843	89.034

Table B: **Ablation studies: GIN vs others. RMSE produced by different methods are shown and GREED with GIN produces the best results.**

Pool functions	CiteSeer (SED)	PubMed (SED)	Amazon (SED)	IMDB (GED)
GREED (Sum)	**0.519**	0.728	**0.495**	**6.734**
GREED-Max	0.795	**0.709**	0.603	52.519
GREED-Mean	0.922	0.732	0.846	52.483
GREED-Attention	0.914	0.797	0.868	130.47

Table C: **Ablation studies: sum-pool vs others. The sum-pool is the best choice among the considered alternatives.**

Impact of sum-pool: To substantiate our choice of the pooling layer, we have performed ablation studies with various pooling functions as replacements for sum-pool. It is clear from Table C that sum-pool is the best choice among the considered alternatives.

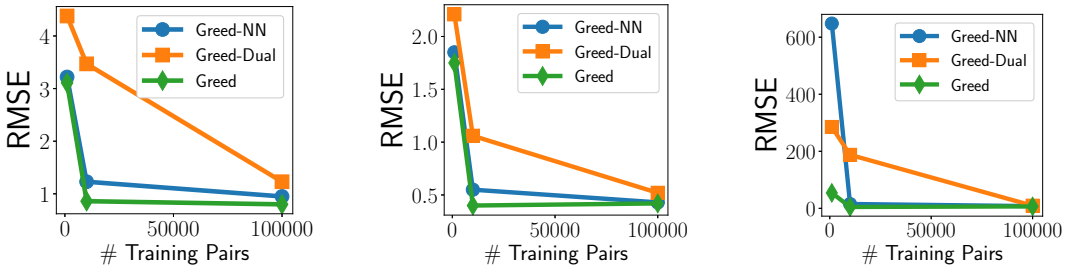
Sum-pool can better distinguish graph sizes better than other aggregation functions such as mean-pool or max-pool. To elaborate, let us consider a graph G_1 that is significantly larger than another graph G_2. In this scenario, the individual coordinates of G_1's embedding can potentially be significantly larger than those of G_2 since in G_1 the summation is being done over a larger set of embeddings. Both mean-pool and max-pool fail to capture the size information as effectively, since the max and the mean operations do not scale with the number of inputs.

Impact of Pre-mlp layer: Table D presents the results. As visible, we do not see any significant difference in performance on average.

Performance of GREED-NN and GREED-Dual on GED: Fig. F presents the results. The trends are similar to what we observed for SED in Sec. 4.5. GREED-NN closes the performance gap with GREED as more training data is provided. Furthermore, GREED-dual is consistently worse that GREED.

RMSE	With Pre-MLP	Without Pre-MLP
AIDS' (SED)	0.51	0.51
Amazon	0.5	**0.39**
CiteSeer	0.52	**0.51**
Cora_ML	**0.64**	0.68
AIDS (GED)	**0.8**	0.85
IMDB (GED)	**6.73**	7.68
Linux (GED)	0.42	**0.41**
Protein	0.52	0.52
PubMed	0.73	0.73

Table D: RMSE of GREED with and without the Pre-mlp layer.



(a) AIDS (b) Linux (c) IMDB

Figure F: Performance of GREED-NN and GREED-Dual on GED. Refer to § 4.5 for details.

Methods	PubMed	Amazon
GREED Retrieval	0.373	6.471
MIP-F2 Alignment	52.8	68.4

Table E: **The average running times in seconds per top-10 query. Our technique is much faster than MIP-F2 alignment.**

H Alignment

In real-world applications of subgraph similarity search, alignments are of interest only for a small number of similar subgraphs. Our framework is intended to serve as a filter to retrieve this small set of similar subgraphs from a large number of candidates. To elaborate, a graph database may contain thousands or millions of graphs (or alternatively, thousands or millions of neighborhoods of a large graph) which need to be inspected for similar subgraphs. A user is typically interested in only a handful of these subgraphs that are highly similar to the query. Since the filtered set is significantly smaller, a non-neural exact algorithm suffices to construct the alignments (Lemma 1 allows us to adapt general cost GED alignment techniques for SED alignment). Computing alignments across the entire database is unnecessary and slows down the query response time.

To substantiate our claim, we show the average running time for answering 10-NN queries. We break up the running time into two components: (i) 10-NN retrieval time by GREED, (ii) exact alignment time using MIP-F2 for the 10-NN neighborhoods retrieved by GREED. We observe that exact alignment by existing methods on the 10-NN neighborhoods completes in reasonable time. In contrast, GENN-A* does not scale on either PubMed or Amazon since it computes alignments across *all* (sub)graphs.

I Visualization

Searching for molecular fragment containment is a routine task in drug discovery [19]. Motivated by this, we show the top-5 matches to an SED query on the AIDS dataset produced by GREED in Fig. E. The query is a functional group (Hydrogen atoms are not represented). GREED is able to extract chemical compounds that contain this molecular fragment (except for ranks 3 and 5, which contain

this group with 1 edit) from around 2000 chemical compounds with varying sizes and structures. This validates the efficacy of GREED at a semantic level.

J Subgraph sampling strategies for SED generalizability

In RWR, we perform fixed length random walks, where the length of a walk is the average diameter size of the queries generated through BFS during training. Next, we merge the walks to form a graph.

In RW, we perform random walks, till the diameter of the resultant graph is the same as the average diameter of the BFS sampled train graphs.

The details of the SHADOW sampler is explained in [40].

K Extension to maximum common subgraph similarity (MCSS)

Besides GED, MCSS is also a popular similarity measure for whole-graph comparison. Except the inductive bias injected through $\mathcal{F}$, all components of GREED is generic for any graph distance function. In this section, we replace $\mathcal{F}$ with an MLP and model MCSS. Table F presents the results. As visible, the trends hold even in this distance function, where GREED outperforms the closest baseline of H^2MN.

Methods	AIDS	Linux	IMDB
GREED	**0.514**	**0.085**	**0.293**
H^2MN	0.652	0.152	0.475

Table F: **RMSE on MCSS.**