Noah: Neural-optimized A* Search Algorithm for Graph Edit Distance Computation

Lei Yang[†], Lei Zou[†,*]
[†]*Peking University, China;*
[*] *National Engineering Laboratory for Big Data Analysis Technology and Application (PKU), China;*
{yang_lei,zoulei}@pku.edu.cn

***Abstract*—Graph Edit Distance (GED) is a classical graph similarity metric that can be tailored to a wide range of applications. However, the exact GED computation is NP-complete, which means it is only feasible for small graphs only. And therefore, approximate GED computation methods are used in most real-world applications. However, traditional practices and end-to-end learning-based methods have their shortcomings when applied for approximate GED computation. The former relies on experience and usually performs not well. The latter is only capable of computing similarity scores between graphs without an actual edit path, which is crucial in specific problems (e.g., Graph Alignment, Semantic Role Labeling). This paper proposes a novel approach Noah, which combines A* search algorithm and graph neural networks to compute approximate GED in a more effective and intelligent way. The combination is mainly reflected in two aspects. First, we learn the estimated cost function $h(\cdot)$ by Graph Path Networks. Pre-training GEDs and corresponding edit paths are also incorporated for training the model, therefore helping optimize the search direction of A* search algorithm. Second, we learn an elastic beam size that can help reduce search size and satisfy various user settings. Experimental results demonstrate the practical effectiveness of our approach on several tasks and suggest that our approach significantly outperforms the state-of-the-art methods.**

***Index Terms*—graph edit distance, A* search algorithm, neural networks**

I. Introduction

Recently, graphs are ubiquitous and have attracted increasing research interest because many data in a wide range of applications can be represented by graphs, such as chemical compounds [1], social networks [2], road networks [3] and semantic web [4]. One of the fundamental problems in such graph-represented applications is graph similarity search (i.e., given a query graph q, finding a set of similar graphs g in a graph database D, such that q is approximately matched with g under some similarity metric). Two classical graph similarity metrics are Graph Edit Distance (GED) [5] and Maximum Common Subgraph (MCS) [6]. Note that the two metrics are inter-related [7], and they are both NP-complete [8]. In this paper, we focus on GED computation.

A. Existing solutions and motivations

The most widely used method for exact GED computation is based on the A* search algorithm [9], which casts it as a path-finding problem and focuses on how to expand the existing search paths. In detail, the algorithm explores the space of all possible mappings between two graphs by means of an ordered tree. Such a search tree is constructed dynamically by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. The further expansion of the search path is determined by a cost function $f(\cdot)$, which can be divided into two parts (i.e., $f(\cdot) = g(\cdot) + h(\cdot)$, where $g(\cdot)$ is the observable cost and $h(\cdot)$ is the estimated cost). Specifically, in each iteration, the search path of minimum cost is selected from the heap of all currently possible paths. In order to guarantee the final result of A* search algorithm to be optimal, the estimated cost $h(\cdot)$ should be lower than, or equal to, the real cost. Based on this, early studies focus on the heuristic functions which can better estimate $h(\cdot)$ [10]–[13], which is a major task in A* search algorithm.

However, since the search space grows exponentially along with more nodes, the exact GED cannot be reliably computed within reasonable time between graphs with more than 16 nodes [14]. To avoid colossal computation costs and satisfy high real-time requirements in many applications, two main categories are proposed in early works. We briefly introduce them and their defects. First, modifications are based on A* search algorithm, such as A*-Beamsearch [15]. Specifically, it limits the size of the heap of A* search algorithm to obtain approximate GEDs in a short time. However, lower bounds based on heuristic functions are not close enough to the ground-truth value of $h(\cdot)$, and parameters of modifications are almost fixed and based on experience. Specifically, the above two problems would bring extra search costs (e.g., the beam size is set high) or miss the optimal results (e.g., the beam size is set low).

Second, end-to-end learning-based methods are applied for graph similarity search [16], [17]. Specifically, they design a network-based function that maps the graph pairs into similarity scores, which turns the GED computation into a *learning* problem. However, this kind of methods might achieve incorrect approximate GED (i.e., the obtained GED is smaller than the exact GED), and therefore could not find an actual edit path from the source graph to the target graph, which is crucial in specific problems such as Graph Alignment [18], Semantic Role Labeling [19], etc. Meanwhile, such methods focus on the evaluation of graph query task (i.e., for each graph in the testing set, we treat it as a query graph, and let the model compute the similarity between the query

graph and every graph in the database), which might not fit for GED computation very well. Because in most cases, the two graphs in the graph pair are not seen before, rather than one of them is in the database.

B. Our approach

We propose a novel approach in this paper to compute approximate GED and address additional tasks (e.g., graph similarity search, graph classification) without the influence of the above disadvantages. Our approach, called **Noah** (i.e., short for **N**eural-**o**ptimized **A*** search algorit**h**m), optimized A* search algorithm through graph neural networks in two aspects. First, the estimated cost function $h(\cdot)$ learned by graph neural networks helps A* search algorithm optimize the search direction. Second, an elastic beam size can help optimize search space and satisfy various user settings. Table I summarizes the characteristics of existing solutions and our approach.

TABLE I
SUMMARY OF EXISTING SOLUTIONS AND NOAH.

Method	Type	Node size	Acc	Edit path
A*	exact	$\leq$16	-	able
A*-Beam	approximate	tens	medium	able
End-to-end	approximate	$\approx$100	low	disable
Noah	approximate	hundreds	high	able

We further propose an extension to GNNs, which we call Graph Path Networks (GPN) for the above neural optimizations. Specifically, we incorporate pre-training GEDs and attention-based edit paths for training the model. Such pre-training information not only enriches training data but also significantly improves the performance of A* search algorithm. On the other hand, we introduce graph alignment information (i.e., node substitution) as the cross-graph information into the training process, which effectively reduces the model size compared to early works. In addition to the above design, we also encode user settings (e.g., permissible error and running time for GED computation) to learn an elastic beam size (i.e., a beam size for each pair of input graphs) for further optimization in A* search algorithm. Through the above main design, Noah can optimize A* search algorithm both in search direction and search space to compute GED more effectively and intelligently. Note that Noah can absolutely find an edit path from the source graph to the target graph as A* search algorithm does.

Our contributions are summarized as follows:

- To the best of our knowledge, we are the first to use neural networks for reinforcing A* search algorithm in GED computation. Our approach, called Noah, optimizes the search direction and search space by predicting the estimated cost function and beam size, respectively.
- We further propose Graph Path Networks for the optimizations. Specifically, it incorporates attention-based pre-training edit path information for training the model, introduces graph alignment information as cross-graph information into the training process, and encodes user settings for learning an elastic beam size.
- We have evaluated our proposal by comparing it with the state-of-the-art baselines on three real-world datasets and a synthetic dataset. Experimental results suggest that our approach significantly outperforms other methods in GED computation metrics and graph similarity metrics across a range of tasks.

The remainder of this paper is organized as follows. Section 2 introduces the preliminaries of our work and reviews A* search algorithm for GED computation. Section 3 poses the workflow of Noah, and Section 4 describes the detailed design of GPN. We evaluate our proposal in Section 5 and discuss related works in Section 6. At last, we conclude in Section 7.

II. PRELIMINARIES AND BACKGROUND

Graph edit distance (GED) defines the dissimilarity of two graphs by the minimum amount of primitive operations needed to transform one graph into the other one. In this section, we first review the terminologies used in this paper, and then review A* search algorithm.

A. Graph Edit Distance

Definition 1. ***Graph.*** *A graph is denoted by a 6-tuple $g = (V, E, L_V, L_E, \Sigma_V, \Sigma_E)$, where V denotes a set of vertices, $E \subseteq V \times V$ a set of (un)directed edges, Σ_V and Σ_E are the label sets of V and E, respectively, and L_V and L_E are label functions that assign labels to vertices and edges, respectively.*

There are six primitive edit operations on graphs: insert/delete a vertex with a label, substitute a/an vertex/edge label, and insert/delete an edge between two vertices. And the edit path is defined as follows.

Definition 2. ***Edit path.*** *Given two graphs g_1 and g_2, there exists a sequence of primitive edit operations to transform g_1 to g_2, such as, $g_1 = g_1^0 \rightarrow g_1^1 \rightarrow \cdots \rightarrow g_1^k = g_2$.*

We may have different operation sequences to transform g_1 to g_2, and each operation sequence can correspond to a node substitution, which is useful information in Graph Alignment.

Definition 3. ***Node substitution.*** *Given two graphs g_1 and g_2, and their vertices $V_1 = \{u_1, \cdots, u_{|(V_1)|}\}$ and $V_2 = \{v_1, \cdots, v_{|(V_2)|}\}$. A node substitution is denoted by $p = \{u_i \rightarrow v_j, \cdots, u_{i'} \rightarrow \varepsilon, \cdots, \varepsilon \rightarrow v_{j'}, \cdots\}$, where $u_i \rightarrow v_j$ denotes the substitution of a vertex u_i by a vertex v_j, $u_{i'} \rightarrow \varepsilon$ denotes the deletion of $u_{i'}$, and $\varepsilon \rightarrow v_{j'}$ denotes the insertion of $v_{j'}$.*

Note that a node substitution only consists of edit operations on vertices, while edit operations on edges can be implied by edit operations on their adjacent vertices.

Definition 4. ***Graph Edit Distance (GED).*** *Given two graphs g_1 and g_2, their GED is defined as the minimum number of primitive operations to transform g_1 to g_2, denoted by $GED(g_1, g_2)$. Note that there might have several edit paths to compute the GED.*

We pose an example of an edit path and its corresponding node substitution in Figure 1. It is also the minimum cost edit

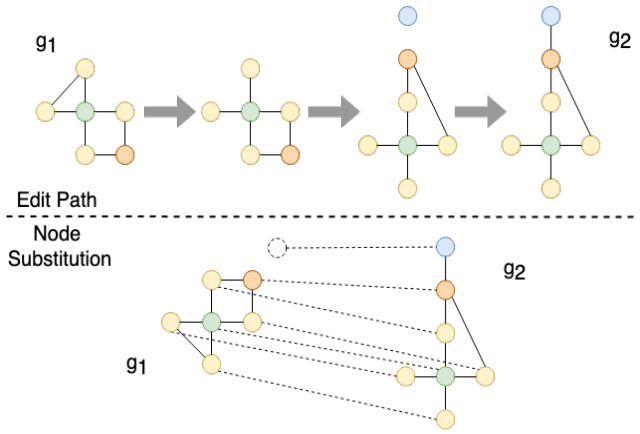


Fig. 1. An example of edit path and its corresponding node substitution.

path between g_1 and g_2. We can transform g_1 to g_2 by an edge deletion, a node insertion, and an edge insertion, sequentially. So that $GED(g_1, g_2) = 3$.

B. Review of A* search algorithm

Since it is impossible and unnecessary to compute all possible edit paths between two graphs (i.e., the source graph g_1 and the target graph g_2), the GED can be solved by A* search algorithm [9].

Algorithm 1: A* algorithm for graph edit distance

Input: Two graphs $g_1 = (V_1, E_1, L_{V_1}, L_{E_1}, \Sigma_{V_1}, \Sigma_{E_1})$ and $g_2 = (V_2, E_2, L_{V_2}, L_{E_2}, \Sigma_{V_2}, \Sigma_{E_2})$, where $V_1 = \{u_1, \cdots, u_{|(V_1)|}\}$ and $V_2 = \{v_1, \cdots, v_{|(V_2)|}\}$

Output: A minimum-cost edit path p_{min} from g_1 to g_2

1 Initialize an empty set OPEN to store edit path ;
2 **for** *each vertex* $\omega \in V_2$ **do**
3 | Insert the substitution $\{u_1 \to \omega\}$ into OPEN ;
4 Insert the deletion $\{u_1 \to \varepsilon\}$ into OPEN ;
5 **while** *OPEN* $\neq \emptyset$ **do**
6 | Remove $p_{min} = argmin_{p \in OPEN}\{g(p) + h(p)\}$ from OPEN;
7 | **if** p_{min} *is a complete edit path* **then**
8 | | **return** p_{min}
9 | **else**
10 | | Let $p_{min} = \{u_1 \to v_{i_1}, \cdots, u_k \to v_{i_k}\}$;
11 | | **if** $k < |V_1|$ **then**
12 | | | **for** *each* $\omega \in V_2 \backslash \{v_{i_1}, \cdots, v_{i_k}\}$ **do**
13 | | | | Insert $p_{min} \cup \{u_{k+1} \to \omega\}$ into OPEN ;
14 | | | Insert $p_{min} \cup \{u_{k+1} \to \varepsilon\}$ into OPEN ;
15 | | **else**
16 | | | Insert $p_{min} \cup \bigcup_{\omega \in V_2 \backslash \{v_{i_1}, \cdots, v_{i_k}\}} \{\varepsilon \to \omega\}$ into OPEN ;
17 **return** $\emptyset$

This algorithm explores the space of all possible mappings between two graphs by means of an ordered tree. Such a search tree is constructed dynamically by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. In order to determine the node which will be used for further expansion in the next iteration, we need a cost function. Formally, for a partial edit path p (i.e., a node in the search tree that can trace back the root), $g(p)$ represents the cost of such a partial edit path accumulated so far, and $h(p)$ represents the estimated cost from the current node to a leaf node. Therefore, the sum $g(p)+h(p)$ can represent a complete solution that transform g_1 to g_2. Obviously, the partial edit path p that minimizes $g(p) + h(p)$ is chosen for further expansion.

Algorithm 1 formally describes A* search algorithm for GED. We assume the vertices of g_1 are processed in the order of $\{u_1, \cdots, u_{|(V_1)|}\}$. All possible edit operations are constructed simultaneously for each vertex, including the removal of the vertex (line 14) and the substitution of the vertex by any unprocessed vertex of g_2 (line 13). If all vertices of g_1 are processed, all unprocessed vertices of g_2 can be inserted into the edit path in a single iteration (line 16). The OPEN set consists of all potential partial edit paths to be considered in the next iteration. In each iteration, we only focus on the path p_{min} which minimizing the cost function $g(p)+h(p)$ (line 6). If p_{min} is a complete path, it is guaranteed to be an optimal one and is returned as the solution (line 8).

Algorithm 2: Observable cost $g(p)$

Input: Two graphs $g_1 = (V_1, E_1, L_{V_1}, L_{E_1}, \Sigma_{V_1}, \Sigma_{E_1})$ and $g_2 = (V_2, E_2, L_{V_2}, L_{E_2}, \Sigma_{V_2}, \Sigma_{E_2})$, an edit path $p = \{u_1 \to v_{i_1}, \cdots, u_k \to v_{i_k}\}$, and it corresponding edit function L_p that assign vertices of g_1 to vertices of g_2

Output: Observable cost $g(p)$

1 Initialize $g(p) \leftarrow 0$;
/* Vertex relabeling */
2 **for** *each vertex* $u \in V_1$ **do**
3 | **if** $L_{V_1}(u) \neq L_{V_2}(L_p(u))$ **then**
4 | | $g(p) \leftarrow g(p) + 1$;
/* Edge deletion or relabeling */
5 **for** *each edge* $(u, u') \in E_1$ **do**
6 | **if** $(L_p(u), L_p(u')) \notin E_2$ *or* $L_{E_1}(\mu, \mu') \neq L_{E_2}(L_p(u), L_p(u'))$ **then**
7 | | $g(p) \leftarrow g(p) + 1$;
/* Edge Insertion */
8 **for** *each edge* $(v, v') \in E_2$ **do**
9 | **if** $(L_p^-(v), L_p^-(v')) \notin E_1$ **then**
10 | | $g(p) \leftarrow g(p) + 1$;
11 **return** $g(p)$

Further more, *observable cost* $g(p)$ for current edit path p can be computed in linear time by Algorithm 2. Specifically, the observable cost $g(p)$ is made up of three parts: vertex relabeling (line 2-4), edge deletion or relabeling (line 5-7) and edge insertion (line 8-10).

C. Estimated cost $h(p)$

Estimated cost $h(p)$ is a significant task in A* search algorithm, because if estimated cost $h(p)$ is lower than, or

equal to, the real cost, an optimal path is guaranteed to be found [9]. Based on this, early works focus on the heuristic function to better estimate $h(p)$, and several kinds of lower bound are proposed. We introduce some typical ones as follows.

1) Label Set-based Lower Bound $\delta^{LS}(\cdot,\cdot)$: Let $g_1\backslash p$ and $g_2\backslash p$ represent the remaining parts of g_1 and g_2, respectively. The label set-based lower bound [20] is

$$\delta^{LS}(g_1\backslash p, g_2\backslash p) = \Phi(V_1(g_1\backslash p), V_2(g_2\backslash p)) + \Phi(E_1(g_1\backslash p), E_2(g_2\backslash p)) \quad (1)$$

where $\Phi(\cdot,\cdot)$ denotes the edit distance between two multisets and $\Phi(S_1,S_2) = max\{|S_1|,|S_2|\} - |S_1 \cap S_2|$ for multisets S_1 or S_2.

2) Star Match-based Lower Bound $\delta^{SM}(\cdot,\cdot)$: The star match-based lower bound is proposed in [21].

Definition 5. ***Star structure.*** *A star structure is a 3-tuple $s = (\nu, L, l)$, where ν denotes the root vertex, L denotes the set of ν's one-hop neighbors, and l denotes a labeling function. Edges exist between ν and any vertex in L and no edge exists among vertices in L.*

A graph g can be mapped to a multiset of star structures, and represented by this multiset as $S(g)$. The edit distance between two star structures can be computed as

$$\lambda^{SM}(s_1,s_2) = \mathbb{1}_{l(s_1)\neq l(s_2)} + ||L(s_1)| - |L(s_2)|| + \Phi(L(s_1), L(s_2)) \quad (2)$$

where s_1 and s_2 are the star structures from $g_1\backslash p$ and $g_2\backslash p$ respectively, $\mathbb{1}_\phi$ is an indicator function that equals 1 if the expression ϕ evaluates true and 0 otherwise. Then, the star match-based lower bound is

$$\delta^{SM}(g_1\backslash p, g_2\backslash p) = \frac{\min_{L_p} \sum_{\nu \in g_1\backslash p} \lambda^{SM}(\nu, L_p(\nu))}{max\{4, [max\{\Delta(g_1\backslash p), \Delta(g_2\backslash p)\} + 1]\}} \quad (3)$$

where L_p denotes the set of all mappings from vertices of $g_1\backslash p$ to vertices of $g_2\backslash p$, and $\Delta(g_1\backslash p)$ and $\Delta(g_2\backslash p)$ denote the maximum vertex degree in $g_1\backslash p$ and $g_2\backslash p$ respectively. Note that ***Hungarian algorithm*** [22] is applied to obtain the minimum cost for star structure mapping in $O(n^3)$ time, where n is the vertex number in involved graphs.

3) Branch Match-based Lower Bound $\delta^{BM}(\cdot,\cdot)$: The branch match-based lower bound is proposed in [10].

Definition 6. ***Branch structure.*** *A branch structure is a 2-tuple $b = (\nu, l_e)$, where ν denotes the root vertex, and l_e denotes the multiset of edge labels adjacent to ν.*

The branch structure is similar to the star structure except for excluding the one-hop neighbor nodes, so that the edit distance between two branch structures can be computed as

$$\lambda^{BM}(b_1,b_2) = \mathbb{1}_{l(b_1)\neq l(b_2)} + \frac{1}{2} \times \Phi(l_e(b_1), l_e(b_2)) \quad (4)$$

Then, the branch match-based lower bound is

$$\delta^{BM}(g_1\backslash p, g_2\backslash p) = \min_{L_p} \sum_{\nu \in g_1\backslash p} \lambda^{BM}(\nu, L_p(\nu)) \quad (5)$$

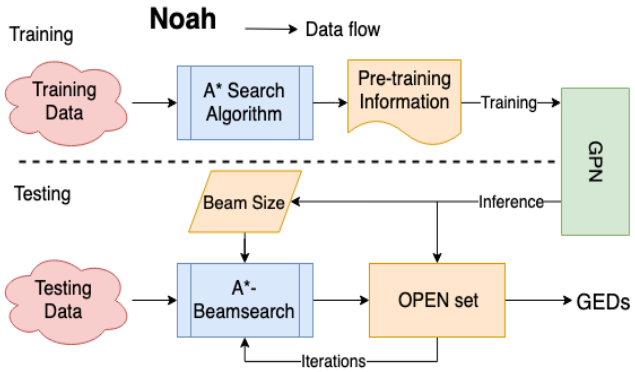


Fig. 2. Workflow of Noah.

where L_p denotes the set of all mappings from vertices of $g_1\backslash p$ to vertices of $g_2\backslash p$. Compared to star match-based lower bound, branch match-based lower bound can be computed in $O(nlogn)$, where n is the vertex number in involved graphs.

III. Overview of Noah

In this paper, in order to optimize A* search algorithm, we introduce GNNs (Graph Neural Networks) in two aspects: predicting the estimated cost $h(p)$ more accurately, and learning variable beam sizes for different graph pairs instead of manually setting based on experiences. Specifically, we design a **N**eural-**o**ptimized **A*** search algori**h**m (called *Noah* for short) in this section.

Figure 2 shows the workflow of our proposal, Noah, consisting two major parts: *Training* and *Testing*. In the training part, training data is computed by exact GED computation methods (e.g., A* search algorithm) to generate pre-training information, which can be used for following training in Graph Path Networks. In the testing part, testing data needs to be computed by A*-Beamsearch. During the initialization of A*-Beamsearch, an elastic beam size is provided by GPN with the input of the graph pair and user settings. Such a combination optimizes the search space by reducing the useless computation of A*-Beamsearch. Another combination is applied in each iteration of A*-Beamsearch. We apply GPN to predict $h(p)$ for *each* edit path in the OPEN set. Such a combination furnishes A* search algorithm with a more accurate estimated cost than traditional lower bounds, which can optimize the search direction by reducing the iterations of A*-Beamsearch. Through the above workflow, Noah is capable of computing GEDs approximately for graph pairs in testing data.

Note that the *training* phase is conducted on small graphs (i.e., graphs with less than 20 nodes), but we can apply the pre-training information to compute GED between large graphs (i.e., graphs with up to hundreds of nodes) in the *testing* phase (see Section V-B3 for details), which confirms the excellent generalization ability.

IV. Graph Path Networks

In order to make Noah effective for GED computation, Graph Path Networks (GPN) is proposed. In this section, we

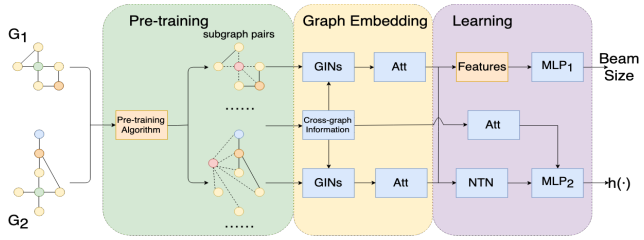


Fig. 3. The overall architecture of Graph Path Networks. The colored areas denote three different modules, in which orange and blue boxes represent algorithms and NN layers, respectively.

present GPN in detail, and the overall architecture of GPN is illustrated in Figure 3.

A. Model overview

GPN is a GNN-based framework which is adaptive to A* search algorithm. It has three main modules (i.e., pre-training module, graph embedding module, and learning module). The pre-training module computes pre-training information (i.e., exact GEDs and graph edit paths), and then generates (sub)graph pairs in the training set. The graph embedding module is designed based on Graph Isomorphism Network (GIN) [23], a state-of-the-art graph neural network which can distinguish graph structures. It first transforms the node of each (sub)graph into a vector, encoding the features and structural properties around each node. In such node-level embeddings, (sub)graph pairs share the same graph neural networks, and cross-graph information is included in the encoded features. After that, it introduces an attention mechanism to obtain the final graph-level embeddings. The learning module is designed for beam size and estimated cost $h(\cdot)$ in A* search algorithm. The following subsections detail the three modules.

B. Pre-training module

Pre-training methods such as BERT [24] and GPT [25] have obtained success in the text domain these years. At the same time, pre-training methods used for graph similarity tasks also achieved better performance than baseline methods without pre-training information [26]. Specifically, pre-training information (i.e., ground-truth GEDs) is incorporated into a supervised loss function

$$\mathcal{L} = \mathbb{E}_{(i,j)\sim\mathcal{D}}(\hat{d_{ij}} - d_{ij})^2, \tag{6}$$

$$= \mathbb{E}_{(i,j)\sim\mathcal{D}}(\|\mathbf{h}_{G_i} - \mathbf{h}_{G_j}\|_2^2 - d_{ij})^2 \tag{7}$$

where $\mathbf{h}_{G_i}$ and $\mathbf{h}_{G_j}$ are the embeddings of the graph pair (i,j), and d_{ij} is their distance. The loss function is to minimize the difference between the predicted and the ground-truth GEDs.

Although the above pre-training information can be useful for some downstream tasks (e.g., graph classification, graph similarity ranking), it is not good enough to guide path-finding in A* search algorithm. First, the estimated cost $h(\cdot)$ in A* search algorithm is the distance between two unprocessed subgraphs, so that the prediction for $h(\cdot)$ would not be accurate if we only use entire graphs for training. Second, the cost function $f(\cdot)$ in A* search algorithm is the sum of observable cost and estimated cost, and therefore, a simple similarity ranking among all candidate edit paths could not guide the path selection in the following iterations of A* search algorithm. In that case, we introduce a better-suited pre-training method for A* search algorithm framework.

1) Take edit paths into consideration: In our pre-training method, the pre-training information is not only the ground-truth GEDs between training graph pairs. We obtain the ground-truth GEDs between training graph pairs with their corresponding node substitutions through A* search algorithm. Note that the obtained node substitution is one of the optimal solutions, and we further decompose it into a series of partial node substitutions and partial edit costs. Specifically, the loss function has been changed as follows:

$$\mathcal{L} = \mathbb{E}_{(i,j)\sim\mathcal{D}} \frac{\sum_{\theta=0}^{|\mathcal{E}|}(\|\mathbf{h}_{G_i}^{\theta} - \mathbf{h}_{G_j}^{\theta}\|_2^2 - d_{ij}^{\theta})^2}{|\mathcal{E}|} \tag{8}$$

where $\mathcal{E}$ is the set of node substitutions between graph pair (i,j), $\mathbf{h}_{G_i}^{\theta}$ and $\mathbf{h}_{G_j}^{\theta}$ are the embeddings of unprocessed parts of the graph pairs after the current node substitution θ, and d_{ij}^{θ} is the corresponding distance between the subgraph pair.

We select the node substitution information rather than edit path information for two reasons. First, the number of edit operations in an edit path equals the GED, while the number of node substitutions equals the number of nodes. The former is significantly larger than the latter, which brings more computation cost. Second, node substitution information better satisfies the requirements of A* search algorithm. Therefore, the actual pre-training information we use is the corresponding node substitutions of the edit paths.

2) Enhanced loss function with attention mechanism: In Equation 8, we assume each node substitution contributes equally to A* search algorithm. However, some critical node substitutions in A* search algorithm might influence the direction of path-finding, which means increasing the weight of these node substitutions can better optimize A* search algorithm. Based on such observation, we apply an attention mechanism between node substitutions as

$$\tilde{\mathbf{h}}_{G_i}^{\theta} = \sum_{k=0}^{\theta} att(\mathbf{h}_{G_i}^{\theta}, \mathbf{h}_{G_i}^{k}) \cdot \mathbf{h}_{G_i}^{k} \tag{9}$$

where $\tilde{\mathbf{h}}_{G_i}^{\theta}$ denotes the improved embedding of unprocessed part of graph G_i after the node substitution θ and $att(\cdot,\cdot)$ is an attention function as

$$att(\mathbf{h}_{G_i}^{\theta}, \mathbf{h}_{G_i}^{k}) = \frac{\alpha_{\theta,k}}{\sum_{k'=0}^{\theta} exp(\alpha_{\theta,k'})}, \tag{10}$$

$$\alpha_{\theta,k} = \omega_1^T \cdot tanh(\mathcal{W}_1 \cdot \mathbf{h}_{G_i}^{k} + \mathcal{W}_2 \cdot \mathbf{h}_{G_i}^{\theta}) \tag{11}$$

where ω_1, $\mathcal{W}_1$ and $\mathcal{W}_2$ are the parameter vector or matrix to learn during the training process.

With the above attention mechanism, we can discover more significant node substitutions in the optimal solution and learn a more appropriate model for predicting $h(\cdot)$. What is more, since previous approaches can hardly solve exact (or approximate) GED computation with plenty of nodes in the graph pairs, learning more significant node substitutions could help A* search algorithm make better path-finding choices. To some extent, the attention mechanism used in pre-

training information works like a kind of metric-based meta-learning [27], which is beneficial for generalization to large graphs.

C. Graph Embedding module

The graph embedding module embeds each (sub)graph into a vector for further measurement based on the selected similarity metric (e.g., GED). We adopt Graph Isomorphism Network (GIN), which has been proven theoretically an example among maximally powerful GNNs [23], in the graph embedding module. For convenience, we take a subgraph pair of (G_1, G_2) as an example in this subsection.

First, the initial nodes can be embedded through a multi-layer perceptron (MLP) if the input features are not one-hot encodings:

$$\mathbf{h}_i^{(0)} = MLP^{(0)}(\mathbf{x}_i), \forall i \in V_1 \tag{12}$$

Then, GIN updates node representations as

$$\mathbf{h}_i^{(k)} = h_{\mathbf{\Theta}}^{(k)}\left((1+\epsilon^{(k)})\cdot \mathbf{h}_i^{(k-1)} + \sum_{j\in\mathcal{N}(i)} \mathbf{h}_j^{(k-1)}\right) \tag{13}$$

where $\mathbf{h}_i$ is the representation of node i, $\mathcal{N}(i)$ is the set of neighbors of node i, $h_{\mathbf{\Theta}}^{(k)}$ denotes a neural network (i.e., a multi-layer perceptron), and ϵ is a learnable parameter or a fixed scalar.

Afterwards, cross-graph information is exploited into node representations update. The cross-graph information we use is the difference between the nodes and their substituted nodes in the paired graphs. Specifically, the node representations update takes into account not only the neighbors of the node but also the graph alignment information. Such information is also generated from the pre-training information of node substitutions. Since we know the node can be matched to one or none (i.e., the vertex deletion) node in the paired graph, we can transform the node substitution information $p = \{u_i \to v_j, \cdots, u_{i'} \to \varepsilon, \cdots, \varepsilon \to v_{j'}, \cdots\}$ into a binary matrix $M \in \mathbb{1}^{|(V_1)|\times|(V_2)|}$, in which each entry equals 0 or 1. If $M_{i,j}$ equals 1, it means node i in graph G_1 and node j in graph G_2 are matched; otherwise, it means not. Therefore, the cross-graph information of node i in graph G_1 can be expressed as

$$\boldsymbol{\mu}_i^{(k)} = \sum_{l\in V_2} \mathbb{1}_{M_{i,l}}(\mathbf{h}_i^{(k-1)} - \mathbf{h}_l^{(k-1)}) \tag{14}$$

where $M_{i,l}$ is the value of the i-th row and the l-th column in the node substitution matrix M, and $\mathbb{1}_\phi$ is an indicator function that equals 1 if the expression ϕ evaluates true and 0 otherwise. Therefore, the node representations are updated as follows

$$\mathbf{h}_i^{(k)} = h_{\mathbf{\Theta}}^{(k)}\left((1+\epsilon^{(k)})\cdot \mathbf{h}_i^{(k-1)} + \sum_{j\in\mathcal{N}(i)} \mathbf{h}_j^{(k-1)} + \boldsymbol{\mu}_i^{(k)}\right) \tag{15}$$

Two main advantages are obtained by including cross-graph information. First, the differences between the nodes and their substituted nodes are amplified through iterations, which makes our model more sensitive to these differences. Second, GIN has a possible limitation that whenever two nodes share the same neighborhood, it may fail to converge to one of the possible solutions. Including cross-graph information helps us distinguish such two nodes by their substituted nodes, so that they are not "mirrored" anymore, and our methods can converge to one of the possible solutions.

Note that the cross-graph information can also be added into node representation without the node substitution matrix. In that case, $\mathbb{1}_{M_{il}}$ is replaced by an attention function (e.g., $att(\mathbf{h}_{G_1}^i, \mathbf{h}_{G_2}^l)$ in Equation 11). However, such cross-graph information induces much computation during training.

Finally, we generate graph-level embedding through a weighted sum of node-level embeddings rather than an unweighted average or sum of node-level embeddings because more important nodes should receive more weights. We should also notice that most GNNs can only embed connected graphs into vectors, while the majority of subgraphs of the graph pairs are unconnected, so that we introduce a hyper-node that connects all nodes and has the same label for every subgraph. And therefore, after the iterations of GNN, the hyper-node can provide the global structural and feature information of the subgraph. Meanwhile, nodes similar to the hyper-node are more critical than those dissimilar. Based on such observation, the graph-level embedding is computed by

$$\mathbf{h}_{G_1} = \sum_{i\in V_1} \sigma(\mathcal{S}_h(\mathbf{h}_i^T, \mathbf{h}_{hyper}))\cdot \mathbf{h}_i \tag{16}$$

where $\mathcal{S}_h(\cdot,\cdot)$ is a vector space similarity metric, like Euclidean similarity or other similarity metrics [28], $\sigma(\cdot)$ is a sigmoid function (i.e., $\sigma(x) = \frac{1}{1+exp(-x)}$) that ensures the similarity metrics in the range $(0, 1)$, and $\mathbf{h}_i$ and $\mathbf{h}_{hyper}$ is the node embeddings for node i and the hyper-node in the subgraph G_1, respectively.

D. Learning module

Through the procedure in the graph embedding module, we obtain the graph-level embeddings for each (sub)graph pair. The learning module aims to utilize these graph embeddings and then help A* search algorithm achieve the ability to predict multiple objectives (i.e., beam size and estimated cost $h(\cdot)$).

1) Predicting the elastic beam size with MLP: We use a multi-layer perceptron to define the cost function for predicting the beam size. Our elastic beam size is predicted for each computation rather than for each iteration for two reasons. First, it costs too much if we predict a beam size for each iteration. Second, during the computation, we could not locate the subgraphs belonging to the optimal solutions. Therefore, only graph embeddings for the complete graphs $\mathbf{h}_{G_1}^{(0)}$ and $\mathbf{h}_{G_2}^{(0)}$ are included in the inputs. Meanwhile, an embedding vector $\boldsymbol{v}_u$ for one-hot encoding user settings (e.g., permissible error and running time for GED computation) is also included in the inputs. The training data is collected by sampling. Formally, we have the following component to infer an elastic beam size based on the user settings.

$$B_s = MLP_1(\mathbf{h}_{G_1}^{(0)}, \mathbf{h}_{G_2}^{(0)}, \boldsymbol{v}_u) \tag{17}$$

2) Improved prediction for $h(\cdot)$: A naive approach to predict the estimated cost $h(\cdot)$ is a standard similarity metric (e.g., Euclidean similarity or Hamming distance [17]) or a multi-layer perceptron as we do in predicting the beam size. However, the prediction for $h(\cdot)$ is much more significant for A* search algorithm than the beam size because it can effectively optimize the search direction and reduce the search space. It is also noted that such simple usage of data representations often leads to insufficient or weak interaction between the graph pair [29]. Therefore, we use Neural Tensor Networks (NTN) to further model the interaction between two graph embeddings:

$$\mathcal{S}(\mathbf{h}_{G_1}, \mathbf{h}_{G_2}) = f(\mathbf{h}_{G_1}^T \mathcal{W}_3^{[1:K]} \mathbf{h}_{G_2} + \mathbf{V} \begin{bmatrix} \mathbf{h}_{G_1} \\ \mathbf{h}_{G_2} \end{bmatrix} + \mathbf{b}) \quad (18)$$

where $\mathcal{W}_3^{[1:K]}$ is a set of (i.e., K refers to the iterations of graph embedding) learned weight tensor, $[\cdot]$ is the concatenation operation, $\mathbf{V}$ is a weight vector, $\mathbf{b}$ is a bias vector, and $f(\cdot)$ is an activation function. Such interaction $\mathcal{S}(\mathbf{h}_{G_1}, \mathbf{h}_{G_2})$ is used to replace the simple similarity metric in Equation 8. Note that normalization is also applied through the higher bound based on the number of nodes and edges for better training.

V. Experiments

A. Experimental Setup

1) Testbed: The single machine we use consists of two 12-core Intel Xeon Gold 6126 CPUs (48 hardware threads in total), a 400 GB Samsung DDR4-2666 DRAM, and four Nvidia Tesla P100 GPUs. All algorithms and evaluations are implemented by *Python*, and the model is written in *PyTorch* and its extension deep learning library, *PyTorch Geometric* [30].

2) Datasets: We use three real-world and one synthetic dataset for evaluation. We introduce them briefly first and then conclude their properties in Table II.

AIDS. The AIDS dataset consists of graphs representing molecular compounds from the AIDS Antiviral Screen Database of Active Compounds. The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. The AIDS dataset has been used by many prior works of graph similarity search or graph classification [16], [31]–[33].

GREC. The GREC dataset consists of graphs representing symbols from architectural and electronic drawings [34]. Ending points, corners, intersections and circles are represented by nodes, and the nodes are connected by undirected edges. We choose the GREC dataset because it has much symmetry and isomorphism, bringing a challenge to GED computation.

IMDB. The IMDB dataset consists of ego-graphs representing movies of different genres from IMDB [35]. For each graph, nodes represent actors/actresses, and there is an edge between them if they appear in the same movie. It represents datasets generated from social networks in specific domains, and it also contains many large graphs.

Synthetic. The synthetic dataset is generated by sampling binomial graphs with n nodes and edge probability p, and then create positive paired graphs by randomly substituting k_p edges from initial graphs with new edges, and negative paired graphs by substituting k_n edges from initial graphs, where $k_p < k_n$.

TABLE II
Detailed information of different datasets

Dataset Type	Graphs	Pairs	Nodes	Partition[1]
AIDS	700	490K	[2, 10]	60% : 20% : 20%
GREC	1100	1.21M	[4, 24]	60% : 20% : 20%
IMDB	1500	2.25M	[7, 89]	60% : 20% : 20%
Synthetic	30000	10K[2]	{20, 100}	60% : 20% : 20%

[1] The partition of datasets is expressed as training set : validation set : testing set.
[2] The synthetic dataset has 10K triples (i.e., initial graphs, positive paired graphs and negative paired graphs).

3) Baselines: Since our approach is a combination of traditional GED computation algorithm (i.e., A* search algorithm) and neural networks, we compare our approach with them.

First, the traditional approximate GED computation algorithms include A*-Beamsearch [15], Hungarian [36], and VJ [37]. Therefore, we use the best result of the above three algorithms as the baseline of traditional approaches. Note that the lower bound we use in A*-Beamsearch is the label set-based lower bound.

Second, recently, two notable end-to-end learning-based models (i.e., SimGNN [16] and GMN [17]) have achieved impressive results on graph similarity search. Hence, we view both of them as the baseline of learning-based approaches.

4) Metrics: The performance of different approaches is evaluated by using two kinds of metrics: (1) GED computation; (2) similarity ranking.

The following metrics are used for *GED computation*: *Accuracy*. It measures the accuracy that the computed GEDs compared to the ground-truth GEDs. *Feasibility*. It measures the ratio that the computed GEDs are feasible (i.e., the computed GEDs are equal to or more than the ground-truth GEDs). *Mean Absolute Error (MAE).* It measures the average absolute difference between the computed GEDs and the ground-truth GEDs, which indicates the computation ability. *Mean Squared Error (MSE).* It measures the average squared difference between the computed GEDs and the ground-truth GEDs, which indicates the computation stability.

The metrics used for similarity ranking are *Spearman's Rank Correlation Coefficient (ρ)* [38], *Kendall's Rank Correlation Coefficient (τ)* [39], and *Precision at k ($p@k$).* The first two metrics measure the matching ratio between the computed ranking results and the real ranking results. The last one measures the matching ratio between the computed top k results and the ground-truth top k results, which focuses on the top k results rather than the global ranking results. Specifically, k we use in the evaluation are 10 and 20.

5) Hyperparameters setting: For model architecture, we set the number of GIN layers to 3, which would cause the final representation of a node to contain information from its 3rd order neighbors. Meanwhile, we use a ReLU nonlinearity on the hidden layers between GIN layers. The output dimensions for the 1st, 2nd, and 3rd GIN layers are 128, 64, and 32, respectively. The initial ϵ is 0, and follow-up ones are learned.

TABLE III
PERFORMANCE COMPARISON FOR PREDICTION ABILITY BETWEEN NOAH AND BASELINES ON THREE DATASETS.

Datasets	Methods \ Metrics	Accuracy	Feasibility	MAE	MSE	ρ	τ	p@10	p@20
AIDS	Traditional[1]	0.303	**1.000**	1.600	4.922	**0.747**	**0.573**	0.804	0.801
	SimGNN	0.201	0.367	1.647	4.720	0.605	0.454	0.788	0.761
	GMN	0.249	0.549	**1.294**	**3.090**	0.686	0.519	0.776	0.731
	Noah	**0.313**	**1.000**	1.542	4.675	0.734	0.560	**0.809**	**0.812**
GREC	Traditional	0.223	**1.000**	1.880	6.919	0.791	0.658	0.516	0.517
	SimGNN	0.154	0.496	2.131	7.758	0.745	0.564	0.148	0.255
	GMN	0.155	0.708	2.096	7.548	0.706	0.533	0.193	0.295
	Noah	**0.275**	**1.000**	**1.790**	**5.845**	**0.807**	**0.671**	0.**562**	**0.552**
IMDB	SimGNN	0.116	0.525	32.489	8839	0.761	0.623	0.347	**0.470**
	GMN	0.167	0.472	39.064	11433	0.807	0.669	0.090	0.130
	Noah	**0.484**	**1.000**	**3.755**	**55.636**	**0.810**	**0.716**	**0.354**	0.405

[1] The traditional baseline is the best result of A*-Beamsearch, Hungarian and VJ.

Since we have introduced cross-graph information from node embeddings to graph embeddings, we set the iteration number K in the NTN layer to 8. The MLPs we use for the learning module are all 2-layers. For training, the number of iterations is set to 5000, and the initial learning rate is 0.001.

B. Tasks, Results and Analysis

In this subsection, we set a series of tasks among different datasets to compare our approach with baselines in several aspects. To briefly summarize, the following three tasks indicate the prediction ability, learning ability, and generalization ability, respectively.

1) Graph Edit Distance Computation and its downstream tasks: MAE and MSE are the basic metrics to examine how good a method can compute GEDs. Meanwhile, GED is also used by its downstream tasks, such as graph isomorphism check [40], graph similarity search [16], graph classification [15] and entity alignment [41]. Therefore, we need other metrics to examine the prediction ability for those downstream tasks. Specifically, checking graph isomorphism (i.e., graph edit distance of 0) requires the accuracy of the prediction results; graph classification task requires the similarity ranking of p@k (to know the tested graphs belong to which known categories); and entity alignment task requires the prediction results feasible (i.e., if the computed GEDs are less than the ground-truth GEDs, it is impossible to exist corresponding edit operations for alignment).

In the training and evaluation process, the training graph pairs are created by training set and validation set, while the testing graph pairs are created by training set, validation set, and testing set. Specifically, training graph pairs = (training set + validation set) $\times$ (training set + validation set), and testing graph pairs = (training set + validation set) $\times$ testing set. The process is in conformance with experiments in early works for similarity search, which can also be viewed as a real-world scenario of graph query [16].

Table III shows the performance for prediction ability on three real-world datasets. Among all four methods, our method, Noah, obtains the best or second performance in all metrics across three datasets. It suggests that our method is not only effective for GED computation metrics but also for graph similarity metrics. It is worth mentioning that traditional methods achieve the best graph similarity metrics on the AIDS dataset, and GMN achieves the best GED computation metrics on the AIDS dataset. We conjecture that graphs in the AIDS dataset have few nodes and simple structural information. We notice that Noah performs much better than other methods in graph similarity metrics on GREC dataset because of symmetry and isomorphism that might not be learned well by end-to-end learning-based methods. We also notice that the gap between GED computation metrics of Noah and other methods becomes much more prominent on the IMDB dataset. Two reasons contribute to such a case. First, graphs in the IMDB dataset are hugely different from each other, causing end-to-end learning-based methods (i.e., SimGNN and GMN) unable to compute GED as traditional methods and combinatorial methods. Second, based on our observation, among three traditional methods, Hungarian and VJ algorithms perform well on the IMDB dataset, while A*-Beamsearch performs well on the other two real-world datasets.

2) Learning how to compute Graph Edit Distance: Note that the graph query task (i.e., for each graph in the testing set, we treat it as a query graph, and let the model compute the similarity between the query graph and every graph in the database) used in Task 1 might be suitable for tasks related to graph similarity search, but not in line with GED computation. Because such a scenario assumes that one graph in the graph pair is what we have seen before, while the reality is that neither of the graphs in the graph pair has been seen. For GED computation, we should pay more attention to the cross-graph information that can be used to learn how to compute GED rather than global information used for graph similarity search.

Therefore, we set the evaluation for this task by only using testing set (i.e., testing graph pairs = testing set $\times$ testing set). In Table IV, Noah performs stably in different testing graph pairs (i.e., comparison between Task 1 and Task 2), while SimGNN and GMN perform much worse than they perform in Table III. Such results demonstrate that Noah can learn from graphs it has met before and learn how to compute GED between two unseen graphs.

TABLE IV
RESULTS OF TESTING PAIRS FOR LEARNING ABILITY ON AIDS.

Method	Acc	Fea	MAE	MSE	ρ	τ	p@10	p@20
SimGNN	0.217	0.506	2.280	13.189	0.192	0.139	0.147	0.204
GMN	0.194	0.492	2.358	13.671	0.152	0.110	0.122	0.153
Noah	**0.319**	**1.000**	**1.501**	**4.483**	**0.748**	**0.578**	**0.748**	**0.758**

3) Train on small graphs, generalize to large graphs: Except for the above prediction ability and learning ability, the generalization ability is also required because one of the advantages for learning-based methods is they can figure out the problem of scalability that traditional algorithms could not cope well with [16], [17]. And therefore, we evaluate the generalization ability by training on small graphs and then testing on large graphs.

However, there is a fundamental problem for generating the testing dataset. As we said above, no currently available algorithms can reliably compute the exact GED for large graphs (i.e., graphs with more than 16 nodes). Meanwhile, the IMDB dataset (i.e., applying the best results of three traditional algorithms as the ground-truth GEDs) introduces noise that might influence the prediction accuracy for all kinds of learning-based methods. For Noah, the influence of such noise is even more, because Noah is designed to learn the search path of A* search algorithm rather than the node substitution of Hungarian or VJ algorithm.

In consideration of the above, we carefully design the synthetic dataset and the task for generalization evaluation. Specifically, as we mentioned in Section V-A, we set the generated graphs with 20 nodes and 100 nodes for training and testing set, respectively, and the edge probability of 0.2. As to graph pairs, we create positive paired graphs and negative paired graphs for each graph in the dataset rather than create graph pairs between all different graphs (i.e., as what we do in three real-world datasets). The positive paired graphs are created by randomly substitute k_p edges from the initial graphs. Because we assume the cost of an edge substitution is 2 in Algorithm 2 (i.e., an edge substitution operation can be divided into an edge deletion and an edge insertion), the real GED between the initial graph and the positive paired graph is $2k_p$. Note that the actual GED between them can be smaller than $2k_p$ due to symmetry and isomorphism. However, the probability of such cases is typically low for large graphs, and for small graphs, we can check such cases by using GED computation methods. The situation is the same for negative paired graphs. Through this design, the ground-truth GEDs for large graphs are also accurate, and the metric for similarity turns to be pairwise similarity (i.e., the accuracy for finding the positive ones). Specifically, k_p and k_n are ranging from 1 to 5, while we guarantee $k_p < k_n$.

TABLE V
RESULTS OF SYNTHETIC DATASET.

Method	Acc	Fea	MAE	MSE	Triplet acc
SimGNN	0.081	0.215	1.130	1.887	0.515
GMN	0.051	0.177	2.579	9.743	0.813
Noah	**0.914**	**1.000**	**0.447**	**0.831**	**0.992**

Table V shows the experimental results of generalization ability between Noah and end-to-end learning-based methods on the Synthetic dataset. Note that the GED computation metrics are used as above, while the similarity metric we used in this experiment is Triplet accuracy because of the way we generate the synthetic dataset. Triplet accuracy can be computed as

$$Acc_{Triplet} = \frac{\sum \mathbb{1}_{GED_{pos} < GED_{neg}}}{N_{Triplets}} \tag{19}$$

where $\mathbb{1}_{\phi}$ is an indicator function that equals 1 if the expression ϕ evaluates true and 0 otherwise, and $N_{Triplets}$ is the total number of triplets. It measures the accuracy for distinguishing positive (or negative) graphs from the testing triplets. The results show that our approach outperforms the others significantly on all five metrics, which indicates that our approach has excellent generalization ability. Benefit from the optimized triplet losses [17], GMN performs much better than SimGNN on the Triplet accuracy metric though it has poorer GED computation performance.

C. Detailed Analysis on Noah

1) Effectiveness of Graph Path Networks: In order to evaluate the effectiveness of GPN, we should design a subgraph testing because the input of the estimated cost $h(p)$ is two subgraphs rather than two complete graphs. We generate the subgraph dataset through the edit paths computed by A* search algorithm, and then use it for evaluation.

First, we compare GPN with three lower bounds (i.e., label set-based, star match-based, and branch match-based) that we introduce in Section II-C, and the comparison metric we use is MAE. Figure 4 shows that Noah outperforms them significantly on all three datasets, which demonstrates the effectiveness of GPN for $h(\cdot)$ prediction. Note that among three lower bounds, branch match-based lower bound outperforms the other two when meeting small graphs (i.e., AIDS dataset), while label set-based lower bound outperforms the other two significantly when meeting large graphs (i.e., GREC and IMDB datasets), so that we choose label set-based lower bound as the default lower bound in A*-Beamsearch.

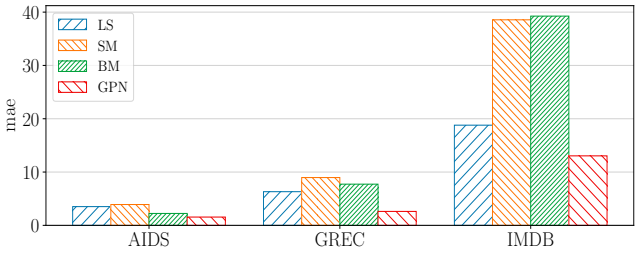


Fig. 4. Comparison between different lower bounds on three datasets.

TABLE VI
RESULTS OF SUBGRAPH TESTING ON AIDS.

Method	Accuracy	MAE	MSE
SimGNN	0.174	1.954	6.457
GMN	0.194	1.739	5.202
GPN	**0.198**	**1.557**	**3.760**

Then, following the thought of combining neural networks with A* search algorithm, we compare the GED computation metrics among SimGNN, GMN, and GPN on the AIDS dataset. The results in Table VI show that GPN performs better than the others in accuracy, MAE, and MSE. Feasibility is unnecessary for subgraph testing in A* search algorithm.

2) Efficiency of estimated cost prediction: Afterwards, we evaluate the efficiency of $h(\cdot)$ prediction by comparing the number of iterations of A* search algorithm between Noah and A*-Beamsearch. The results in Figure 5 show that Noah can significantly reduce the number of iterations in all three real-world datasets. Noted that it reduces more iterations in more massive graphs, and the reduced ratios of three datasets are 22.4%, 34.4%, and 44.5%, respectively.

Furthermore, reducing the number of iterations of A* search algorithm is not enough to prove the efficiency because one inference of our model takes about 100× than function-based lower bounds (i.e., several milliseconds compared to tens of microseconds). Therefore, we expand the beam size of A*-Beamsearch to 10× bigger than our approach to promise they are compared equally (i.e., they all have 20 seconds for search) and then compare their performance on the IMDB dataset. The results in Table VII show that Noah can still perform better than traditional lower bounds on nearly every metric except for p@10, even when they have the same time for search in A* search algorithm.

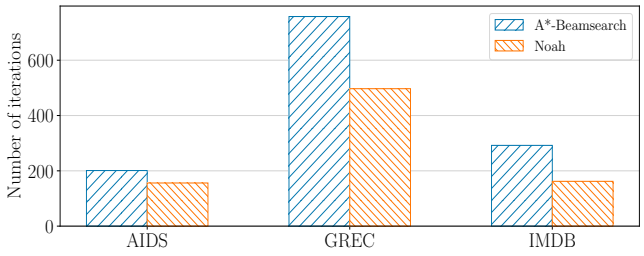
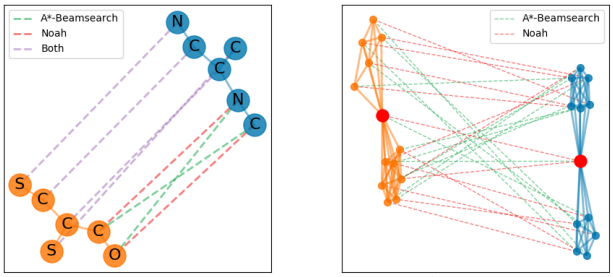


Fig. 5. Comparison on iterations of A* search algorithm.

TABLE VII
RESULTS OF EQUAL COMPARISON ON IMDB.

Method	Acc	Fea	MAE	MSE	ρ	τ	p@10	p@20
A*-Beam	0.276	**1.000**	4.943	263.331	0.773	0.685	**0.360**	0.373
Noah	**0.484**	**1.000**	**3.755**	**55.636**	**0.810**	**0.716**	0.354	**0.405**

D. Case Studies and Insights



(a) AIDS (b) IMDB

Fig. 6. Case studies on AIDS and IMDB dataset.

In this subsection, we first visualize some experimental results as case studies. Then, from these visualizations, we analyze how Noah could obtain such effective performance in GED computation. At last, we propose some insights on combinatorial methods like Noah and our future works.

Figure 6(a) and Figure 6(b) visualize the edit path of sample graph pairs on AIDS dataset and IMDB dataset, respectively. In the left subfigure, A*-Beamsearch (i.e., green dotted lines) pairs the vertex of label "C" in the source graph with the vertex of label "C" in the target graph, while Noah pairs the vertex of label "C" in the source graph with the vertex of label "N" in the target graph. Through different mappings of vertices, Noah only needs two edit operations (i.e., two vertex substitutions), while A*-Beamsearch needs three edit operations (i.e., an edge deletion, an edge insertion, and a vertex relabeling). In the right subfigure, since graphs in IMDB dataset are all ego-networks, Noah can identify egos (i.e., big and red nodes) and then pair them in the edit path, which might significantly reduce edit operations in the edit path. The results indicate that Noah can focus more on the graph structure information than traditional lower bounds in A* search algorithm.

Figure 7 visualize another case study on the GREC dataset, which compare graph similarity through GED with end-to-end learning-based methods (i.e., SimGNN and GMN). The graphs in the first column are the source graphs, and those in the other four columns are the target graph. The results suggest that Noah could recognize more symmetry and isomorphism components, which are common in the GREC dataset, than end-to-end learning-based methods.

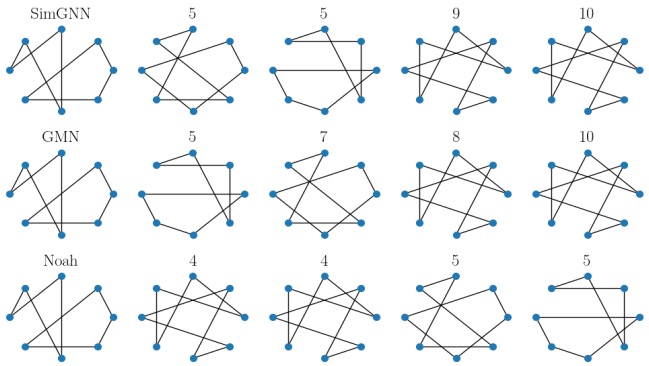


Fig. 7. A case study on GREC dataset.

Insights. Through a series of experiments and case studies, we summarize the following advantages of combinatorial methods like Noah. First, compared with traditional heuristic search algorithms, it does not require manual settings (i.e., beam size based on experience, heuristic $h(\cdot)$ functions), but automatically learns from data. Second, end-to-end learning-based methods belong to transductive learning, which has poor generalization ability, while the combinatorial methods can generalize across different sizes of graphs and different tasks. Meanwhile, it obtains not only GED scores (or similarity scores) learned by graph neural networks, but also edit path which is necessary for downstream tasks. Third, the combinatorial strategy can better absorb the strengths of the two components, which results in accurate prediction capability, especially for complex graph structure information.

Future works. Noah is not perfect and has the following two main limitations. First, though its performance on several datasets proves its efficiency, it is still a heavy computation method compared to end-to-end learning-based methods. Therefore, there is still a need to make it light-weight, such as pruning neural networks, further improving the prediction accuracy, and only inference for "key" iterations. Second, our method exploits GIN with its expressive power for graph isomorphism testing, and therefore inherits its limitations.

Though such limitation is nearly eliminated by cross-graph information, as we mentioned in Section IV-C, it is still possible to occur (i.e., two nodes that share the same neighborhood both have no substituted nodes). Therefore, we might still need a specific kind of GNN for GED computation.

VI. Related Works

We summarize the research directions related to our work.

Graph Edit Distance Computation. Except for exact GED computation and approximate GED computation methods we have introduced in Section I, there is also another kind of method for approximate GED computation, such as Hungarian [36] and VJ [37]. This kind of method is based on an (optimal) fast bipartite optimization procedure mapping nodes and their local structure of one graph to nodes and their local structure of another graph. Hungarian and VJ make use of Munkres' algorithm and Volgenant and Jonker algorithm for finding an optimal match between the sets of local structure, respectively. Originally, these algorithms have been proposed to solve the assignment problem in polynomial time, which can be generalized to compute approximate GEDs.

Deep Learning for Graph Similarity Search. Recent years have witnessed the success of deep learning in modeling complex data structures (i.e., graph) and relationships. In specific, GCN (Graph Convolutional Networks) [42], [43] and GraphSAGE [44] are proposed to learn node representations and then provide node-level embedding for graphs. Furthermore, aggregation based methods such as simple average or weighted average [45] and attention mechanism [46], [47] are exploited to generate graph-level embedding. For specific use in graph similarity search, SimGNN [16] and GMN [17] both introduce cross-graph information into their models. These studies mainly abstract graph information to features that deep learning models can learn, and then solve practical problems, such as node or graph classification, rather than our current task (i.e., GED computation).

Machine Learning for Complicated Algorithms and Heuristics. More recently, learning-based approaches have been proved to have the potential to yield complicated algorithms and heuristics by learning from massive data because they can effectively detect useful patterns and leverage modeling capacity, which may escape algorithm designers. Specifically, learning-based approaches have been introduced in classic NP-hard problems, such as Satisfiability, Traveling Salesman, Maximum Cut, Minimum Vertex Cover [48]–[51]. Also, strategies learned by reinforcement learning in the game GO [52] and Atari games [53] have been proved effective than ever before. Furthermore, combinatorial optimization with machine learning and traditional search algorithms takes the advantages of both. It outperforms recent deep learning works on some specific problems, such as Maximal Independent Set [54] and Personalized Route Recommendation [55].

VII. Conclusion

In this paper, we propose a novel approach called Noah, which combines A* search algorithm and Graph Path Networks (GPN) we proposed for Graph Edit Distance computation. First, we learn the estimated cost function $h(\cdot)$ by GPN. Attention-based pre-training information and cross-graph information are incorporated for training the model, and therefore optimize the search direction of A* algorithm. Second, we learn an elastic beam size that can help reduce search size and satisfy various user settings. Experimental results suggest that our approach significantly outperforms other methods in graph similarity metrics and GED computation metrics across a range of tasks.

Acknowledgment

This work was supported by Scientific and technological innovation 2030 - new generation of artificial intelligence major project 2020AAA0108505 and NSFC under grant 61932001, 61961130390, U20A20174. This work was also partially supported by Beijing Academy of Artificial Intelligence (BAAI) and Key Research and Development Program of Hubei Province (No. 2020BAB026). The corresponding author of this work is Lei Zou (zoulei@pku.edu.cn).

References

[1] D. Bonchev, *Chemical graph theory: introduction and fundamentals*. CRC Press, 1991, vol. 1.
[2] D. J. Watts, P. S. Dodds, and M. E. Newman, "Identity and search in social networks," *science*, vol. 296, no. 5571, pp. 1302–1305, 2002.
[3] J. Beasley and N. Christofides, "Vehicle routing with a sparse feasibility graph," *European Journal of Operational Research*, vol. 98, no. 3, pp. 499–511, 1997.
[4] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gstore: answering sparql queries via subgraph matching," *Proceedings of the VLDB Endowment*, vol. 4, no. 8, pp. 482–493, 2011.
[5] H. Bunke, "What is the distance between graphs," *Bulletin of the EATCS 20*, pp. 35–39, 1983.
[6] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern recognition letters*, vol. 19, no. 3-4, pp. 255–259, 1998.
[7] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.
[8] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.
[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.
[10] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Efficient graph similarity search over large graph databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 964–978, 2014.
[11] L. Chang, X. Feng, X. Lin, L. Qin, and W. Zhang, "Efficient graph edit distance computation and verification via anchor-aware lower bound estimation," *arXiv preprint arXiv:1709.06810*, 2017.
[12] D. B. Blumenthal and J. Gamper, "Exact computation of graph edit distance for uniform and non-uniform metric edit costs," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2017, pp. 211–221.
[13] K. Riesen, S. Emmenegger, and H. Bunke, "A novel software toolkit for graph edit distance computation," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2013, pp. 142–151.
[14] D. B. Blumenthal and J. Gamper, "On the exact computation of the graph edit distance," *Pattern Recognition Letters*, 2018.
[15] M. Neuhaus, K. Riesen, and H. Bunke, "Fast suboptimal algorithms for the computation of graph edit distance," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2006, pp. 163–172.

[16] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "Simgnn: A neural network approach to fast graph similarity computation," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 384–392.

[17] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3835–3845.

[18] D. Koutra, H. Tong, and D. Lubensky, "Big-align: Fast bipartite graph alignment," in *2013 IEEE International Conference on Data Mining (ICDM)*, 2013.

[19] H. Fürstenau and M. Lapata, "Graph alignment for semi-supervised semantic role labeling," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2009.

[20] D. B. Blumenthal and J. Gamper, "Exact computation of graph edit distance for uniform and non-uniform metric edit costs," in *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, ser. Lecture Notes in Computer Science, vol. 10310, 2017, pp. 211–221.

[21] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 25–36, 2009.

[22] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logs*, vol. 2, no. 1, pp. 83–98, 1955.

[23] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[24] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," 2019.

[25] R. Alec, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[26] Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang, "Unsupervised inductive graph-level representation learning via graph-graph proximity," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pp. 1988–1994.

[27] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, 2016, pp. 3630–3638.

[28] K. Yang, X. Ding, Y. Zhang, L. Chen, B. Zheng, and Y. Gao, "Distributed similarity queries in metric spaces," *Data Sci. Eng.*, vol. 4, no. 2, pp. 93–108, 2019.

[29] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 926–934.

[30] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[31] M. Neuhaus and H. Bunke, *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific, 2007.

[32] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.

[33] K. Riesen and H. Bunke, "IAM graph database repository for graph based pattern recognition and machine learning," in *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5342. Springer, 2008, pp. 287–297.

[34] P. Dosch and E. Valveny, "Report on the second symbol recognition contest," in *Graphics Recognition. Ten years review and future perspectives. Proc. 6th Int. Workshop on Graphics Recognition (GREC'05)*, ser. LNCS 3926. Springer, 2005, pp. 381–397.

[35] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. ACM, 2015, pp. 1365–1374.

[36] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision computing*, vol. 27, no. 7, pp. 950–959, 2009.

[37] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding up graph edit distance computation through fast bipartite matching," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2011, pp. 102–111.

[38] C. Spearman, "The proof and measurement of association between two things." *American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

[39] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, 1938.

[40] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[41] M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege, "Deep graph matching consensus," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[42] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 3844–3852.

[43] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[44] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 1024–1034.

[45] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2224–2232.

[46] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," 2018.

[47] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[48] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[49] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, 2015, pp. 2692–2700.

[50] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2017, pp. 6348–6358.

[51] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[52] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[53] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[54] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, 2018, pp. 537–546.

[55] J. Wang, N. Wu, W. X. Zhao, F. Peng, and X. Lin, "Empowering a* search algorithms with neural networks for personalized route recommendation," in *the 25th ACM SIGKDD International Conference*, 2019.