NED: An Inter-Graph Node Metric Based On Edit Distance

Haohan Zhu*
Facebook Inc.
zhuhaohan@fb.com

Xianrui Meng*
Apple Inc.
xmeng@apple.com

George Kollios
Boston University
gkollios@cs.bu.edu

ABSTRACT

Node similarity is fundamental in graph analytics. However, node similarity between nodes in different graphs (inter-graph nodes) has not received enough attention yet. The inter-graph node similarity is important in learning a new graph based on the knowledge extracted from an existing graph (transfer learning on graphs) and has applications in biological, communication, and social networks. In this paper, we propose a novel distance function for measuring inter-graph node similarity with edit distance, called **NED**. In NED, two nodes are compared according to their local neighborhood topologies which are represented as unordered k-adjacent trees, without relying on any extra information. Due to the hardness of computing tree edit distance on unordered trees which is NP-Complete, we propose a modified tree edit distance, called **TED***, for comparing unordered and unlabeled k-adjacent trees. TED* is a metric distance, as the original tree edit distance, but more importantly, TED* is polynomially computable. As a metric distance, NED admits efficient indexing, provides interpretable results, and shows to perform better than existing approaches on a number of data analysis tasks, including graph de-anonymization. Finally, the efficiency and effectiveness of NED are empirically demonstrated using real-world graphs.

1. INTRODUCTION

Node similarity is an essential building block for many graph analysis applications and is frequently used to develop more complex graph data mining algorithms. Applications requiring node similarity include node classification, similar node retrieval, and topological pattern matching.

In particular, node similarity measures between nodes in different graphs (inter-graph nodes) can have many important applications including transfer learning across networks and graph de-anonymization [7]. An example comes from biological networks. It has been recognized that the topological structure of a node (neighborhood) in a biological network (e.g., a PPI network) is related to the functional and biological properties of the node [6]. Furthermore, with the increasing production of new biological data and networks, there is an increasing need to find nodes in these new networks that have similar topological structures (via similarity search) with nodes in already analyzed and explored networks [5]. Notice that, additional information on the nodes can be used to enhance the distance function that we compute using the network structure.

Another application comes from communication networks. Consider a set of IP communication graphs from different days or different networks that have been collected and only one of these networks has been analyzed [7]. For example, nodes in one network may have been classified into different roles based on their positions in the graph. The question is how to use this information to classify nodes from the other networks without building new classifiers (i.e., across-network classification) [7].

Finally, another important application of inter-graph node similarity is to use it for de-anonymization. As an example, given an anonymous social network and certain non-anonymized information in the same domain, we could compare pairwise nodes to re-identify nodes in the anonymous social network by using the structural information from the non-anonymized corresponding graphs [7, 22].

In recent years, many node similarity measures have been proposed but most of them work only for nodes in the same graph (intra-graph). Examples include SimRank [8], SimRank variants [30, 2, 10], random walks with restart [27], influence-based methods [13], set matching methods [29, 11, 12] and so on. Unfortunately these methods cannot be used to measure the similarity between inter-graph nodes.

The existing methods that can be used for inter-graph node similarity [1, 3, 7, 4] have their own issues. OddBall [1] and NetSimile [3] only consider the features in the ego-net (instant neighbors) which limits the neighborhood information. On the other hand, although ReFeX [7] and HITS-based similarity [4] consider larger neighborhood, they are not metrics and the absolute distance values between different pairs of nodes are not comparable. Furthermore, their distance values are not easy to interpret.

The objective of this paper is to develop a distance function for comparing two nodes in different graphs, where the distance function is both metric and efficient to compute. The first technical challenge is how to select the "signatures" to represent nodes based only on the neighborhood topological structures and at the same time include as much information as possible. The second challenge is how to

*Work done while at Boston University.

Proceedings of the VLDB Endowment, Vol. 10, No. 6

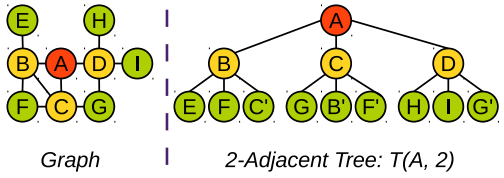


Figure 1: K-Adjacent Tree

compare the "signatures" efficiently and precisely in order to express the similarity between two nodes. The third challenge is how to guarantee that important properties for the distance function such as "metric" and "interpretable" can be satisfied.

In this paper, we propose a novel distance function for measuring inter-graph node similarity with edit distance, called **NED**. In our measure, two inter-graph nodes are compared according to their neighborhood topological structures which are represented as unordered and unlabeled k-adjacent trees. In particular, the NED between a pair of inter-graph nodes is equal to a modified tree edit distance between the pair of corresponding unordered and unlabeled k-adjacent trees. The modified tree edit distance is called **TED*** which is also proposed in this paper. We introduce TED* because the problem of computing the original tree edit distance on unordered trees is NP-Complete. TED* is not only polynomially computable, but it also preserves all metric properties as the original tree edit distance does. TED* is empirically demonstrated to be efficient and effective in comparing trees. Since TED* is a metric edit distance, NED can admit efficient indexing and provides results that are interpretable. According to our case study in the experiments, NED performs very well on graph de-anonymization applications.

Overall, in this paper we make the following contributions:

- We propose a polynomially computable and metric distance function, NED, to measure the similarity between inter-graph nodes.
- We propose a modified tree edit distance, TED*, to compare unordered trees, where TED* is both metric and polynomially computable.
- We show that TED* can be a good approximation to the original tree edit distance for unordered trees.
- We show that NED can admit efficient indexing for similarity retrieval on nodes due to the metricity.
- We use the graph de-anonymization application as a case study to show that NED performs very well in efficiency and precision for inter-graph node similarity.

2. NEIGHBORHOOD TOPOLOGIES

2.1 Unordered K-Adjacent Tree

We first introduce the unlabeled unordered k-adjacent tree that we use to represent the neighborhood topological structure of each node. The k-adjacent tree was firstly proposed by Wang et al. [28] and for completeness, we include the definition here:

DEFINITION 1. *The adjacent tree $T(v)$ of a vertex v in graph $G(V, E)$ is the breadth-first search tree starting from vertex v. The k-adjacent tree $T(v, k)$ of a vertex v in graph $G(V, E)$ is the top k-level subtree of $T(v)$.*

The difference between the k-adjacent tree in [28] and in this paper is that we do not sort the children of each node based on their labels. Thus, the k-adjacent tree in this paper is an unordered and unlabeled tree structure.

An example of a k-adjacent tree is illustrated in Figure 1. For a given node in a graph, its k-adjacent tree can be retrieved deterministically by using breadth first search. In this paper we use this tree to represent the topological neighborhood of a node that reflects its "signature" in the graph. In this paper, we consider undirected graphs for simplicity. However, the k-adjacent tree can also be extracted from directed graphs. E.g., one node can have one inbound k-adjacent tree and one outbound k-adjacent tree. Similarly, our distance metric between inter-graph nodes can be applied to directed graphs as well by considering both inbound and outbound trees.

2.2 Isomorphism Complexity

In this section, we discuss why the neighborhood tree is chosen to represent the node "signature" rather than the neighborhood subgraph.

In general, the identity property for measuring inter-graph node similarity should be defined as follows:

DEFINITION 2. *For two nodes u and v, $u \equiv v$ if and only if $\delta(u, v) = 0$.*

The above definition indicates that two nodes are equivalent if and only if the distance between two nodes is 0. When comparing two nodes only based on their neighborhood topological structures, the isomorphism between the "signatures" of two nodes is the most precise way to measure the node equivalency. Therefore, the identity property can be written as in Definition 3.

DEFINITION 3. *For two nodes u and v, $S(u) \simeq S(v)$ if and only if $\delta(u, v) = 0$.*

In Definition 3, $S(u)$ and $S(v)$ are the "signatures" of nodes u and v respectively and $S(u) \simeq S(v)$ denotes that the two "signatures" are isomorphic. Hence, the computability of the metric distance functions for comparing inter-graph nodes cannot be easier than the computability of the "signature" isomorphism.

In our NED, we choose the neighborhood k-adjacent trees as the "signatures" of nodes. Then the rooted tree isomorphism is used to represent the node equivalency. Thus, it is possible to construct a polynomially computable distance function which satisfies the metric properties.

However, if the k-hop neighborhood subgraphs are the "signatures" of nodes, the restricted graph isomorphism (in Lemma 1) will be used to represent the node equivalency. It guarantees that to satisfy the identity in Definition 2 and 3, the problem of computing node similarity with k-hop neighborhood subgraphs is as hard as graph isomorphism, which belongs to class NP, but not known to belong to class P. Due to space limitation, we leave the proof in [34].

LEMMA 1. *Given two nodes $u \in G_u$, $v \in G_v$ and a value k, $G_s(u, k)$ and $G_s(v, k)$ are k-hop neighborhood subgraphs of u and v. $G_s(u, k)$ and $G_s(v, k)$ are restricted isomorphic if $G_s(u, k) \simeq G_s(v, k)$ and $v = f(u)$, where f is the bijective node mapping for the isomorphism. Then if there exists a distance function $\delta(u, v)$ which satisfies Definition 3, the computation of distance function δ is at least as hard as the graph isomorphism computation.*

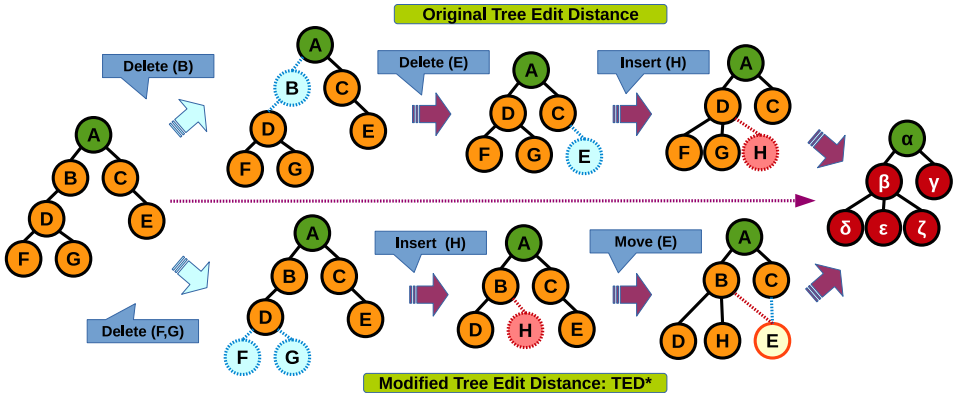


Figure 2: TED* vs Tree Edit Distance

For example, the graph edit distance is a metric between two graphs which can be a distance function to compare inter-graph nodes based on k-hop neighborhood subgraphs. However, as mentioned in Lemma 1, the graph edit distance is not polynomial-time computable. Actually, the computation of graph edit distance is known to be NP-Hard [31]. There are several approximated algorithms for computing graph edit distance [18, 21, 31]. However, although the approximations make the computation efficient the bounds are very loose and make the resulting distance not viable.

Notice that Jin et al. [10] proposed a set of axiomatic role similarity properties for intra-graph node measures. The major difference between axiomatic properties and metric properties is that in the set of axiomatic role similarity properties, the identity is only verified in one direction. Namely, if two nodes are automorphic, the distance is 0. Whereas, if the distance is 0, two nodes may not be automorphic. The reason of one way direction verification is because the node automorphism is chosen to represent the node equivalency and the graph automorphism problem is also not known if it belongs to class P.

Therefore, in this paper, we choose the tree structure to represent the neighborhood topology of a node. Actually, any spanning tree can be a representation of a node. We adopt the k-adjacent tree because it can be deterministically extracted and we show in Section 7.1, the k-adjacent tree can capture the neighborhood topological information very well.

3. NODE SIMILARITY

3.1 TED*

As explained in Section 2.2, the neighborhood tree is a suitable "signature" for comparing inter-graph nodes using only topological information. Before introducing node similarity, we should define how to compare the neighborhood trees efficiently and also preserve the metric properties.

The tree edit distance [26] is a well-defined and popular metric for tree structures. For a given pair of trees, the tree edit distance is the minimal number of edit operations which convert one tree into the other. The edit operations in the original tree edit distance include: 1) Insert a node; 2) Delete a node; and 3) Rename a node. For unlabeled trees, there is no rename operation.

Although the ordered tree edit distance can be calculated in $O(n^3)$ [19], the computation of unordered tree edit distance has been proved to belong to NP-Complete [33], and it is even MaxSNP-Hard [32]. The neighborhood trees in this paper are unordered and unlabeled k-adjacent trees. Therefore, we propose a novel modified tree edit distance called TED* which still satisfies the metric properties like the original tree edit distance but it can be polynomially computable.

The difference between the modified tree edit distance TED* and the original tree edit distance TED is that we impose an additional restriction: no edit operation can change the depth of an existing tree node. The reason is that, we view the depth of a neighbor node, which represents the closeness between the neighbor node and the root node, as an important property of this node in the neighborhood topology. Therefore, two nodes with different depths should not be matched to the same entity. Thus, we have a different set of edit operations for TED* which is defined on this new set of edit operations.

3.2 Edit Operations in TED*

In the original tree edit distance, when inserting a node between an existing node n and its parent, it increases the depth of node n as well as the depths of all the descendants of node n. Similarly, when deleting a node which has descendants, it decreases the depths of all the descendants. Since in TED* we do not want to change the depth of any node, we should not allow these operations. Instead, we need another set of edit operations as follows:

- Insert a leaf node
- Delete a leaf node
- Move a node at the same level

To clarify, "Move a node at the same level" means changing an existing node's parent to another. The new parent node should be in the same level as the previous one. We introduce this "move" edit operation because only operating on leaf nodes will significantly increase the number of operations when comparing trees. The above 3 modified edit operations do not change the depth of any existing node. Also after any edit operation, the tree structure is preserved.

Figure 2 shows an example of the difference between TED and our modified tree edit distance TED*. When converting the tree T_A to the tree T_α, the traditional tree edit distance requires 3 edit operations: delete node B, delete node E and insert node H. TED* requires 4 edit operations: delete node F, delete node G, insert node H and move node E.

Here we define the TED* as follows:

DEFINITION 4. *Given two trees T_1 and T_2, a series of edit operations $E = \{e_1, ...e_n\}$ is valid denoted as E_v, if T_1 can be converted into an isomorphic tree of T_2 by applying the edit operations in E. Then $\delta_T(T_1, T_2) = \min |E|$, $\forall E_v$.*

where each edit operation e_i belongs to the set of edit operations defined above. $|E|$ is the number of edit operations in E and δ_T is the TED* distance proposed in this paper.

In this paper, number of edit operations is considered in TED* which means each edit operation in TED* has a unit cost. However, TED* can also be extended to a weighted version. Notice that, for the same pair of trees, distance TED* may be smaller or larger than TED. In Section 6, we analyze the differences among TED*, TED and the graph edit distance GED in more details, where the TED* can be used to provide an upper-bound for GED on trees and weighted TED* can provide an upper-bound for TED.

3.3 NED

Here, we introduce NED, the inter-graph node similarity with edit distance. Let u and v be two nodes from two graphs G_u and G_v respectively. For a given parameter k, two k-adjacent trees $T(u,k)$ and $T(v,k)$ of nodes u and v can be extracted separately. Then, by applying the modified tree edit distance TED* on the pair of two k-adjacent trees, we can get the similarity between the pair of nodes.

Denote δ^k as the distance function NED between two nodes with parameter k and denote δ_T as TED*. Then we have, for a parameter k,

$$\delta^k(u,v) = \delta_T(T(u,k), T(v,k)) \quad (1)$$

For completeness, we present the metric properties for node similarity here. The NED $\delta^k(u,v)$ is a metric, if and only if it satisfies all 4 metric properties: non-negativity, symmetry, identity and triangular inequality. Namely, for any node u, v and w, and parameter k, the following holds:

1) $\delta^k(u,v) \geq 0$
2) $\delta^k(u,v) = \delta^k(v,u)$
3) $\delta^k(u,v) = 0$, iff $T(u,k) \simeq T(v,k)$
4) $\delta^k(u,v) \leq \delta^k(u,w) + \delta^k(w,v)$

where $T(u,k) \simeq T(v,k)$ denotes the tree $T(u,k)$ is isomorphic to the tree $T(v,k)$ where the node u is mapped to the node v. Notice that, the modified tree edit distance TED* also satisfies all 4 metric properties which means like NED is a metric for nodes, TED* is a metric for trees.

4. TED* COMPUTATION

Since NED between nodes is equal to TED* between trees, the most important part of computing NED is to calculate TED*. In this section, we introduce the algorithm to compute TED* between a pair of k-adjacent trees. Before illustrating the algorithm, we introduce some definitions.

DEFINITION 5. *Let $L_i(u)$ be the i-th level of the k-adjacent tree $T(u,k)$, where $L_i(u) = \{n | n \in T(u,k), d(n,u) = i\}$ and $d(n,u)$ is the depth of node n in $T(u,k)$.*

In Definition 5, the i-th level $L_i(u)$ includes the nodes with depths of i in the k-adjacent tree $T(u,k)$. Similarly in k-adjacent tree $T(v,k)$, there exists the i-th level $L_i(v)$. The algorithm compares two k-adjacent trees $T(u,k)$ and $T(v,k)$ bottom-up and level by level. First, the algorithm compares and matches the two bottom levels $L_k(u)$ and $L_k(v)$. Then the next levels $L_{k-1}(u)$ and $L_{k-1}(v)$ are compared and matched. So on and so forth until two roots.

$T(u,k)$	k-adjacent tree of node u
$L_i(u)$	ith-level of k-adjacent tree of node u
$C(n)$	Canonization label of node n
$x \sqsubset y$	Node x is a child of node y
P_i	Padding cost for the level i
M_i	Matching cost for the level i
G_i^2	Complete bipartite graph in the level i
$w(x,y)$	Edge weight in G_i^2 between x and y
$m(G_i^2)$	Minimal cost for G_i^2 matching
$f_i : f_i(x) = y$	Node mapping function for G_i^2 matching

Table 1: Notation Summarization for TED* Algorithm

When comparing and matching two levels, we use *canonization labels* of nodes from the corresponding levels. The canonization label is defined as follows:

DEFINITION 6. *Let $C(n)$ be the canonization label of node n, $C(n) \in \mathbb{Z}_{\geq 0}$. The canonization label $C(n)$ is assigned based on the subtree of node n. Two nodes u and v have the same canonization labels $C(u) = C(v)$, if and only if the two subtrees of nodes u and v are isomorphic.*

Canonization labels are different from the original node labels and depend on the subtrees only. In the following algorithm, $x \sqsubset y$ means x is an instant child of node y.

The notations used in the algorithm are listed in Table 1.

Algorithm 1: Algorithm for TED* Computation

Input: Tree $T(u,k)$ and Tree $T(v,k)$
Output: $\delta_T(T(u,k), T(v,k))$

for $i \leftarrow k$ **to** 1 **do**
  Calculate padding cost: $P_i = \big||L_i(u)| - |L_i(v)|\big|$;
  if $|L_i(u)| < |L_i(v)|$ **then**
    Pad P_i nodes to $L_i(u)$;
  else if $|L_i(u)| > |L_i(v)|$ **then**
    Pad P_i nodes to $L_i(v)$;
  foreach $n \in L_i(u) \cup L_i(v)$ **do**
    Get node canonization: $C(n)$;
  foreach (x,y), *where* $x \in L_i(u)$ & $y \in L_i(v)$ **do**
    Get collection $S(x) = (\ C(x') \mid \forall x' \sqsubset x\ )$;
    Get collection $S(y) = (\ C(y') \mid \forall y' \sqsubset y\ )$;
    $w(x,y) = |S(x) \setminus S(y)| + |S(y) \setminus S(x)|$;
  Construct bipartite graph G_i^2 with weights $w(x,y)$;
  Get cost $m(G_i^2)$ for minimal matching of G_i^2;
  Calculate matching cost $M_i = (m(G_i^2) - P_{i+1})\ /\ 2$;
  if $|L_i(u)| < |L_i(v)|$ **then**
    foreach $x \in L_i(u)$ **do** $C(x) = C(f_i(x))$;
  else
    foreach $y \in L_i(v)$ **do** $C(y) = C(f_i^{-1}(y))$;
Return $\delta_T(T(u,k), T(v,k)) = \sum_{i=1}^{k}(P_i + M_i)$;

4.1 Algorithmic Overview

The overview of TED* computation is in Algorithm 1. The inputs of the algorithm are two k-adjacent trees and the output is TED* distance.

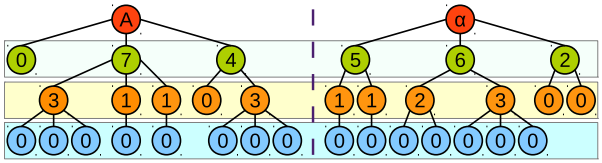


Figure 3: Node Canonization

In the algorithm, there are two types of costs: padding and matching. Since the algorithm runs bottom-up level by level, for each level i, there exists a local padding cost P_i and a local matching cost M_i. The TED* distance is the summation of padding and matching costs from all levels.

Actually, as explained in the following sections, there exists one-to-one mapping from padding and matching to edit operations defined in Section 3.2. The padding cost represents the number of edit operations of 1) Inserting a leaf node and 2) Deleting a leaf node, and the matching cost is the number of edit operations of 3) Moving a node at the same level.

To compute the padding and matching costs, we use 6 steps in each level: node padding (line 2-6 in Algorithm 1), node canonization (line 7-8), complete weighted bipartite graph construction (line 9-13), weighted bipartite graph matching (line 14), matching cost calculation (line 15) and node re-canonization (line 16-19). Next, we describe those 6 steps in details.

4.2 Node Padding

The padding cost is the size difference between two corresponding levels. Let $L_i(u)$ and $L_i(v)$ be the corresponding levels. The difference between the number of nodes in $L_i(u)$ and the number of nodes in $L_i(v)$ is the padding cost:

$$P_i = \big||L_i(u)| - |L_i(v)|\big| \tag{2}$$

There is no edit operation that can change the number of nodes except inserting a leaf node or deleting a leaf node. More importantly, inserting a leaf node and deleting a leaf node are symmetric. If transforming from level $L_i(u)$ to $L_i(v)$ needs inserting several leaf nodes, on the other hand, transforming from level $L_i(v)$ to $L_i(u)$ needs deleting several leaf nodes. Without loss of generality, we use padding costs to represent the number of inserting / deleting leaf nodes for one way transformation.

4.3 Node Canonization

After node padding, we assign the canonization labels to all the nodes in the corresponding levels $L_i(u)$ and $L_i(v)$. Namely, $\forall\ n \in L_i(u) \cup L_i(v)$, we assign the canonization label $C(n)$ to node n.

Based on Definition 6, two nodes x and y have the same canonization labels $C(x) = C(y)$, if and only if the two subtrees of nodes x and y are isomorphic. All the leaf nodes (new inserted ones and existing ones) should have the same canonization labels. However, there is no need to check the full subtrees of two nodes to decide whether they should be assigned the same canonization label or not. We can use the children's canonization labels to decide whether two nodes have the same subtrees or not. Let $S(x)$ be the collection of the canonization labels of all the children of x, i.e.

Definition 7. *$S(x) = (C(x'_1)...C(x'_{|x|}))$, where $x'_i \sqsubset x$ for $1 \le i \le |x|$ and $|x|$ is the total number of node x's children.*

The collection of the canonization labels may maintain duplicate labels, since two children may have the same canonization labels. Also the canonization labels in a collection can be lexicographically ordered. Therefore, we have the following Lemma 2.

Lemma 2. *$C(x) = C(y)$ iff $S(x) \equiv S(y)$.*

Note that the equivalence $\equiv$ denotes that two collections $S(x)$ and $S(y)$ contain exactly the same elements.

Algorithm 2: Node Canonization

Input: Two levels $L_i(u)$ and $L_i(v)$
Output: $C(n)$, $\forall\ n \in L_i(u) \cup L_i(v)$

```
1  Queue q is lexicographically ordered;
2  foreach n ∈ L_i(u) ∪ L_i(v) do
3      Get collection S(n) = ( C(n') | ∀n' ⊏ n );
4      q ← S(n)
5  Pop the first element in q as q_0 = S(x);
6  C(x) = 0;
7  for i ← 1 to |L_i(u)| + |L_i(v)| − 1 do
8      if q_i = S(y) ≡ q_{i−1} = S(x) then
9          C(y) = C(x)
10     else
11         C(y) = C(x) + 1
```

Algorithm 2 illustrates the linear process of node canonization which utilizes the lexicographical orders on canonization label collections. Figure 3 shows an example of node canonization level by level.

The node canonization process guarantees that the nodes with the same canonization label in the same level must have isomorphic subtrees. The matching cost computation can only rely on the instant children without checking all the descendants. Such process makes levels independent and the overall algorithm can be polynomially computable.

4.4 Bipartite Graph Construction

To calculate the matching cost, we need to construct a complete weighted bipartite graph and compute the minimum bipartite graph matching.

The weighted bipartite graph G_i^2 is a virtual graph. The two node sets of the bipartite graph are the corresponding levels from two k-adjacent trees: $L_i(u)$ and $L_i(v)$. The bipartite graph construction is after the node padding. Therefore $L_i(u)$ and $L_i(v)$ must have the same number of nodes.

G_i^2 is a complete weighted bipartite graph which means that for every node pair (x, y) where $x \in L_i(u)$ and $y \in L_i(v)$, there exists a virtual weighted edge. The key component of the bipartite graph construction is to assign the weights to all virtual edges.

In the complete bipartite graph G_i^2, the weight of each edge is decided by the children's canonization labels of two nodes that the edge connects. For two nodes x and y, let the children's canonization label collections be $S(x)$ and $S(y)$. We denote $S(x) \setminus S(y)$ as the difference between collections $S(x)$ and $S(y)$. The weight $w(x, y)$ is the size of the symmetric difference between the collections $S(x)$ and $S(y)$, i.e.

Algorithm 3: Bipartite Graph Construction

Input: Two levels $L_i(u)$ and $L_i(v)$
Output: Bipartite graph G_i^2

foreach (x, y), *where* $x \in L_i(u)$ & $y \in L_i(v)$ **do**
  Get collection $S(x) = (\ C(x') \mid \forall x' \sqsubset x\ )$;
  if x *is a padding node* **then** $S(x) = \emptyset$;
  Get collection $S(y) = (\ C(y') \mid \forall y' \sqsubset y\ )$;
  if y *is a padding node* **then** $S(y) = \emptyset$;
  $w(x, y) = |S(x) \setminus S(y)| + |S(y) \setminus S(x)|$;
    (" $\setminus$ " is collection difference)
  $G_i^2 \leftarrow w(x, y)$;

$w(x, y) = |S(x) \setminus S(y)| + |S(y) \setminus S(x)|$. Figure 4 gives an example of constructing the complete weighted bipartite graph and Algorithm 3 shows the process of constructing the bipartite graph in details.

Since TED* computation goes from the bottom level to the root level, when constructing the bipartite graph G_i^2 by using $L_i(u)$ and $L_i(v)$, the level $L_{i+1}(u)$ and level $L_{i+1}(v)$ have already been matched which means the canonization label collections in $L_{i+1}(u)$ must be the same as in $L_{i+1}(v)$. Notice that, the padded nodes are not connected to any parent node to avoid replicated cost.

The weight between a pair of nodes indicates the number of "moving a node at the same level" edit operations needed. By calculating the minimum matching, we can get the minimum number of moving edit operations accordingly.

4.5 Bipartite Graph Matching

The minimal bipartite graph matching is to find a bijective mapping function f_i for G_i^2, where $f_i(x) = y$, $x \in L_i(u)$ and $y \in L_i(v)$. The bijective mapping function is to minimize the summation of weights from all nodes in $L_i(u)$ to the corresponding nodes in $L_i(v)$. Let $m(G_i^2)$ be the minimal cost of the matching. Then, we have:

$$f_i: \quad m(G_i^2) = Min \sum_{\forall x \in L_i(u)} w(x, f_i(x)) \tag{3}$$

In this paper, we use the Hungarian algorithm to solve the matching problem. In the bipartite graph matching process, we get the bijective mapping function f_i and the minimal cost $m(G_i^2)$ which we will use in the next step: matching cost calculation. Notice that, the minimal cost for bipartite graph matching is not the matching cost in TED* algorithm.

4.6 Matching Cost Calculation

The matching cost M_i represents the minimal number of "moving a node at the same level" edit operations needed to convert level $L_i(u)$ to level $L_i(v)$. Therefore, the matching cost M_i is calculated based on the bipartite graph matching cost $m(G_i^2)$ in the same levels and the padding cost P_{i+1} from the previous levels as shown in Lemma 3.

LEMMA 3. $M_i = (m(G_i^2) - P_{i+1})/2$

PROOF. There are two situations for calculating M_i: $P_{i+1} = 0$ and $P_{i+1} \neq 0$

If there is no padding node in level $L_{i+1}(u)$ and level $L_{i+1}(v)$, the canonization label collection in $L_{i+1}(u)$ must be the same as collection in $L_{i+1}(v)$. Then for any z in the canonization label collection, let $C(n_u) = z$, where $n_u \in L_{i+1}(u)$ and $C(n_v) = z$, where $n_v \in L_{i+1}(v)$. Assume n_u is the child of node $x \in L_i(u)$ and n_v is the child of node $y \in L_i(v)$. If $f_i(x) = y$, where f_i is the bijective mapping function in the matching, then node n_u and node n_v will not generate any disagreement cost in bipartite graph matching. Otherwise, the pair of nodes n_u and n_v will cost 2 in the bipartite graph matching, since x is matching to some other node other than y and y is also matching to some other node. However, only one "moving a node at the same level" edit operation is needed to correct the disagreement, for example, move node n_v from y to $f_i(x)$. Thus, the matching cost should be equal to $m(G_i^2)/2$.

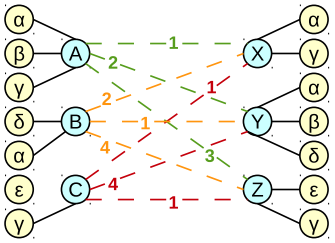


Figure 4: Complete Weighted Bipartite Graph

If there exists padding nodes, then we can always pad the nodes to the optimal parent which will not generate any matching cost again. When constructing the complete bipartite graph G_i^2, one padding node in level $L_{i+1}(u)$ or level $L_{i+1}(v)$ will generate 1 disagreement cost because the padding node is not connected to any node in $L_i(v)$ or $L_i(u)$. Therefore, the number of "moving a node at the same level" operations should be $(m(G_i^2) - P_{i+1})/2$. □

4.7 Node Re-Canonization

The last step for each level is node re-canonization. This step ensures that for each level, only the children information is needed to perform all 6 steps. Let f_i be the bijective matching mapping function from $L_i(u)$ to $L_i(v)$. Then, $\forall\ x \in L_i(u)$, $f_i(x) = y \in L_i(v)$. Symmetrically, $\forall\ y \in L_i(v)$, $f_i^{-1}(y) = x \in L_i(u)$.

In node re-canonization, without loss of generality, we always re-canonize based on the level without padding nodes. Therefore, the node re-canonization criteria is as follows:

- If $|L_i(u)| < |L_i(v)|$, $\forall\ x \in L_i(u)$, $C(x) \Leftarrow C(f_i(x))$
- If $|L_i(u)| \leq |L_i(v)|$, $\forall\ y \in L_i(v)$, $C(y) \Leftarrow C(f_i^{-1}(y))$

After the node re-canonization, the canonization labels of nodes in level $L_i(u)$ are the same as the canonization labels of nodes in level $L_i(v)$. Then we can proceed to the next pair of levels: $L_{i-1}(u)$ and $L_{i-1}(v)$.

5. PROOFS AND ANALYSIS

5.1 Correctness Proof

LEMMA 4. *Algorithm 1 correctly returns the TED* distance defined in Definition 4.*

Before the proof, we rewrite the formula to calculate TED* distance as follows:

$$\delta_T = \sum_{i=1}^{k}(P_i + M_i) = \sum_{i=2}^{k} P_i/2 + \sum_{i=1}^{k-1} m(G_i^2)/2 \tag{4}$$

PROOF. Let x and z be two trees. To prove $\delta_T(x,z)$ returns minimal number of edit operations transforming from tree x to tree z, we could prove that for any tree y, $\delta_T(x,z) \leq \delta_T(x,y) + \delta_T(y,z)$. Because all edit operations preserve tree structures.

In order to prove $\delta_T(x,z) \leq \delta_T(x,y) + \delta_T(y,z)$, we can prove that the following inequalities hold for each level i:

$$P_i^{xz} \leq P_i^{xy} + P_i^{yz} \tag{5}$$

$$m(G_i^2)^{xz} \leq m(G_i^2)^{xy} + m(G_i^2)^{yz} \tag{6}$$

First of all, let $L_i(x)$ be the ith level of k-adjacent tree extracted from node x. Similarly, $L_i(y)$ and $L_i(z)$ are the levels for nodes y and z respectively. According to Algorithm 1, $P_i = \big||L_i(x)| - |L_i(z)|\big|$. Then we have:

$$\begin{aligned} P_i^{xz} &= \big||L_i(x)| - |L_i(z)|\big| \\ &= \big|(|L_i(x)| - |L_i(y)|) - (|L_i(z)| - |L_i(y)|)\big| \\ &\leq \big||L_i(x)| - |L_i(y)|\big| + \big||L_i(z)| - |L_i(y)|\big| \\ &= P_i^{xy} + P_i^{yz} \end{aligned}$$

Therefore, Inequality (5) holds.

Next, let f be the bijective mapping function from level $L_i(x)$ to level $L_i(z)$ which satisfies the minimal bipartite graph matching. Similarly, let g and h be the bijective mapping functions from level $L_i(x)$ to level $L_i(y)$ and from level $L_i(y)$ to level $L_i(z)$. Then, for any node $\alpha \in L_i(x)$, we have $f(\alpha) \in L_i(z)$. Also, for any node $\alpha \in L_i(x)$, we have $g(\alpha) \in L_i(y)$ and for any node $\beta \in L_i(y)$, we have $h(\beta) \in L_i(z)$.

According to Algorithm 1, we can rewrite the minimal cost for bipartite graph matching $m(G_i^2)^{xz}$, $m(G_i^2)^{xy}$ and $m(G_i^2)^{yz}$ as follows:

$$\begin{cases} m(G_i^2)^{xz} = \sum w(\alpha, f(\alpha)) \\ m(G_i^2)^{xy} = \sum w(\alpha, g(\alpha)) \\ m(G_i^2)^{yz} = \sum w(\beta, h(\beta)) \end{cases} \tag{7}$$

In equations above, the weights in three bipartite graphs satisfy the triangular inequality, because the weights are calculated using $w(x,y) = |S(x) \setminus S(y)| + |S(y) \setminus S(x)|$. Therefore, we have:

$$w(\alpha,\gamma) \leq w(\alpha,\beta) + w(\beta,\gamma) \tag{8}$$

Then we prove the inequality (6). Since f, g and h are all bijective mapping functions, so we know for any node $\alpha \in L_i(x)$, both $f(\alpha) \in L_i(z)$ and $h(g(\alpha)) \in L_i(z)$ hold. Then according to Inequality (8), we have:

$$w(\alpha, h(g(\alpha))) \leq w(\alpha, g(\alpha)) + w(g(\alpha), h(g(\alpha))) \tag{9}$$

Because $m(G_i^2)^{xz} = \sum w(\alpha, f(\alpha))$ is the minimal bipartite graph matching cost, so we have:

$$\begin{aligned} m(G_i^2)^{xz} &= \sum w(\alpha, f(\alpha)) \\ &\leq \sum w(\alpha, h(g(\alpha))) \\ &\leq \sum w(\alpha, g(\alpha)) + \sum w(g(\alpha), h(g(\alpha))) \\ &= m(G_i^2)^{xy} + m(G_i^2)^{yz} \end{aligned}$$

Thus, the Inequality (6) is proved.

Because both Inequality (5) and Inequality (6) hold for each level, we could prove that TED* returns minimal number of edit operations. Because there is no intermediate tree structure can reduce the number of edit operations. □

5.2 Metric Proof

The TED* is a metric, if and only if it satisfies all metric properties: non-negativity, symmetry, identity and triangular inequality.

By definition, it is straightforward that the TED* satisfies non-negativity, symmetry, and triangular inequality. Because all edit operations have cost of 1, the distance has to be non-negative. Because all the edit operations are invertible, the distance is symmetric. Since we prove that TED* returns minimal number of edit operations in Section 5.1, the TED* has to satisfy triangular inequality.

In the following part, we prove that TED* satisfies the identity property as well:

LEMMA 5. *$\delta_T(T(x,k), T(y,k)) = 0$, iff $T(x,k) \simeq T(y,k)$*

PROOF. If $\delta_T(T(x,k), T(y,k)) = 0$, there is no edit operation needed to convert $T(x,k)$ to an isomorphic tree of $T(y,k)$. Then the two trees are isomorphic.

If two k-adjacent trees $T(x,k)$ and $T(y,k)$ are isomorphic, there exists a bijective mapping function f from all nodes in tree $T(x,k)$ to the nodes in $T(y,k)$. Then, in each level, number of nodes from two trees should be the same. Then the padding cost is 0 for each level. Also in each level, the bijective mapping function f makes the bipartite graph matching to return 0. Therefore the matching cost is 0. Thus, for a pair of isomorphic trees, TED* must be 0. □

5.3 Complexity Analysis

The TED* computation in Algorithm 1 is executed level by level and includes 6 steps sequentially. We analyze the time complexity of the algorithm level by level. Let m and n be the number of nodes in the corresponding levels from two trees. Without losing generality, assume $m \geq n$.

The node padding can be executed in $O(m-n)$ time and the node canonization can be calculated in $O((m+n)\log(m+n))$ time in our Algorithm 2. The bipartite graph construction needs $O(mn)$ time to generate all weights for a completed bipartite graph. The most time consuming part is the bipartite graph matching. We use the improved Hungarian algorithm to solve the bipartite graph matching problem with time complexity $O(m^3)$ which is state-of-art. The matching cost calculation can be executed in constant time and node re-canonization is in $O(m)$.

Clearly, the time complexity is dominant by the bipartite graph matching part which cost $O(m^3)$. Notice that, m is the number of nodes at one level. Therefore, the overall time complexity of computing TED* should be $O(km^3)$, where k is the number of levels of the tree. Indeed, for the real-world applications, we demonstrate that there is no need for a large parameter k in Section 7.3. For small k, the number of nodes per level is also not large. Therefore, the TED* can be computed efficiently in practice.

5.4 Parameter K and Monotonicity

In NED, there is only one parameter k which represents how many levels of neighbors should be considered in the comparison. There exists a monotonicity property on the distance and the parameter k in NED:

LEMMA 6. $\delta_T(T(u,x),T(v,x)) \leq \delta_T(T(u,y),T(v,y))$, $\forall x,y>0$ *and* $x \leq y$

PROOF. The proof of Lemma 6 is based on the procedures in Algorithm 1. In Lemma 6 , x and y are total number of levels for k-adjacent trees. According to Equation 4 and the non-negativity property in TED*.

Then to prove Lemma 6, we could prove the following inequality instead:

$$\sum_{i=1}^{x}(P_i^y + M_i^y) \geq \sum_{i=1}^{x}(P_i^x + M_i^x) \tag{10}$$

According to the algorithm, for the levels from 1 to x, $P_i^y = P_i^x$, since the top x levels of $T(u,y)$ and $T(v,y)$ should have the same topological structures as $T(u,x)$ and $T(v,x)$ respectively. Meanwhile, we have $M_x^y \geq M_x^x$, because at the xth level, the children of nodes in $T(u,y)$ and $T(v,y)$ may have different canonization labels, but the nodes in $T(u,x)$ and $T(v,x)$ are all leaf nodes. So the matching cost between $T(u,x)$ and $T(v,x)$ at the xth level should not be larger than the matching cost between $T(u,y)$ and $T(v,y)$ at the xth level. For all the levels above the xth level, the matching cost for two distances should be the same. □

The monotonicity property is useful for picking the parameter k for specific tasks as follows: the node similarity, NED, for a smaller parameter k is a lower bound of NED for a larger parameter k. Then, for nearest neighbor similarity node queries, increasing k may reduce the number of "equal" nearest neighbor nodes in the result set. For top similarity node ranking, increasing k may break the ties in the rank. In Section 7.3, we show how the monotonicity property affects the query quality using real world datasets.

6. TED*, TED AND GED

This section briefly discusses the differences among TED*, tree edit distance (TED) and graph edit distance (GED).

The major difference among TED*, TED and GED is edit operations. Basically, since the edit operations are different, the edit distances are not comparable. The edit operations for TED* and TED keep the tree structures, whereas, the operations for GED do not.

One edit operation for TED may be translated into a series of edit operations for GED. However, every edit operation in TED* can be exactly represented as two edit operations in GED: "Inserting a leaf node" in TED* is equivalent to inserting an isolated node and inserting an edge in GED. "Deleting a leaf node" in TED* is equivalent to deleting an edge and deleting an isolated node in GED. "Moving a node in the same level" in TED* can be represented as deleting an edge and inserting an edge in GED. Therefore, TED* can be a bound for GED on tree structures as follows:

$$\delta_{GED}(t_1,t_2) \leq 2 \times \delta_{TED*}(t_1,t_2) \tag{11}$$

However, TED* can be smaller or larger than TED. Since TED allows to insert or delete intermediate node (with parent and children) in a tree and such operations may be translated into a series of edit operations in TED*. Then TED may be smaller than TED*. On the other hand, TED* allows to move a node in the same level but TED does not. So TED has to use a series of edit operations to mock one "Moving a node in the same level" operation in TED*. Therefore, TED may be larger than TED*.

Table 2: Datasets Summary

Dataset	# Nodes	# Edges
CA Road (CAR)	1,965,206	2,766,607
PA Road (PAR)	1,088,092	1,541,898
Amazon (AMZN)	334,863	925,872
DBLP (DBLP)	317,080	1,049,866
Gnutella (GNU)	62,586	147,892
Pretty Good Privacy (PGP)	10,680	24,316

Whereas, we can propose a weighted TED* that can be an upper-bound of TED, for example:

$$\delta_{T(W)} = \sum_{i=1}^{k}(w_i^1 * P_i + w_i^2 * M_i)$$

If $w_i^1 = 1$ and $w_i^2 = 4i$, $\delta_{T(W)}$ is an upper-bound for TED.

Since weighted TED* is not the major discussion in this paper and due to the space limitation, we leave the upper-bound and metric proof in [34].

7. EXPERIMENTS

In this section, we empirically evaluate the efficiency and effectiveness of the inter-graph node similarity. All experiments are conducted on a computing node with 2.9GHz CPU and 32GB RAM running 64-bit CentOS 7 operating system. All experiments are implemented in Java.

In particular, we evaluate TED* and NED over 5 aspects: 1) Compare the efficiency and distance values of TED* against TED and GED; 2) Evaluate the performance of TED* with different sizes of trees; 3) Analyze the effects of parameter k on NED; 4) Compare the computation and nearest neighbor query performance of NED with HITS-based similarity [4] (HITS) and Feature-based similarity [7] (Feature) which are state-of-art inter-graph node similarity measurements; and 5) Provide a case study of graph de-anonymization. Notice that, the Feature-based similarity here is ReFeX [7]. As mention in [7], the means and sums of degrees for each level are recursively collected as features. The details of these state-of-art inter-graph node similarity measurements are introduced in Section 8.

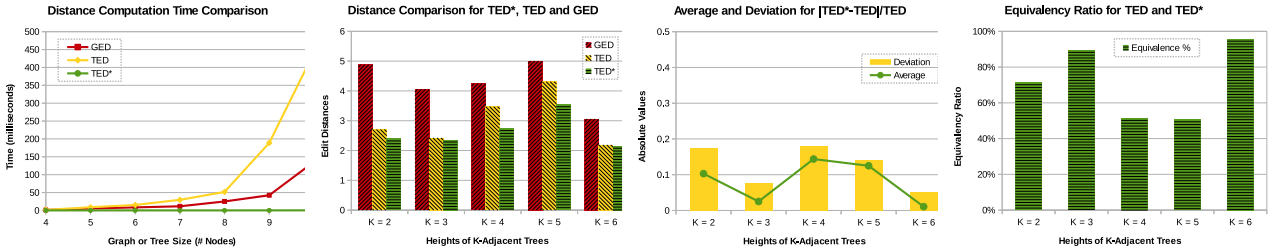
The datasets used in the experiments are real-world graphs that come from the KONECT [14] and SNAP [15] datasets. Table 2 summarizes the statistical information and the abbreviations used in this section.

All the distances in the experiments are computed between pairs of nodes from two different graphs. For each graph, we random sample the nodes uniformly and treat all the nodes equally.

7.1 TED*, TED and GED Comparison

In this section, we check the efficiency of TED* and compare how close is the computed distance to TED and GED on the same unordered trees.

TED* is not only polynomially computable but is also very efficient. On the other hand, computing the exact values of TED and GED is extremely expensive since the computations of TED and GED are NP-Complete problems. The most widely used method for computing exact values is the A*-based algorithm. However, this method can only deal with small topological structures with merely up to 10-12 nodes. Whereas, TED* is able to compare unordered trees up to hundred nodes in milliseconds as shown



(a) Computation Time (b) Distance Values (c) Distance Difference (d) Equivalency Ratio

Figure 5: Comparisons among TED*, TED and GED

in Section 7.2. In this experiment, we extract 20 nodes from (CAR) graph and 20 nodes from (PAR) graph. Totally 400 pairs of nodes are used to calculate TED*, TED and GED.

In Figure 5a, we show the computation time of TED*, exact TED, and exact GED. Actually, among 400 pairs of nodes, more than half of pairs cannot return any result for TED and GED due to the exponential number of branches in A* algorithm.

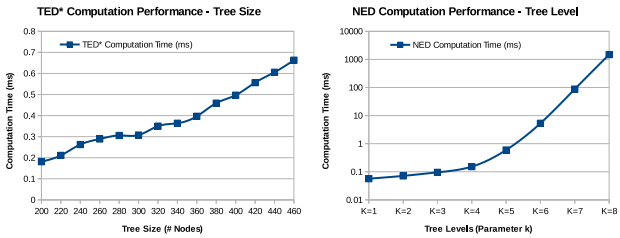
In Figure 5b, we show the differences of distance values in TED*, TED, and GED for unordered trees. In this experiment, we would like to demonstrate two claims: (1) TED* could be an upper bound for half GED because there is a "1-to-2" mapping from the edit operations of TED* to the edit operations of GED as illustrated in Section 6. (2) TED* is pretty similar to TED even though TED* may be smaller or larger than TED as shown Section 6. According to the Figure 5b, TED* is slightly smaller than TED only because for the computable pairs of trees (small trees) there are more "move a node" operations in TED* which should be series of edit operations in TED.

In Figure 5c, we further show the differences between TED* and TED in details as a complement for Figure 5b that shows TED* is pretty similar to TED. Figure 5c shows the average and standard deviation of the relative errors between TED* and TED. The difference is calculated by

$$|TED - TED^*|/TED$$

The average is from 0.04 to 0.14 and the deviation is below 0.2. This means that in most cases the TED and TED* values are almost the same, since the minimal cost for the edit distances is 1 (one edit operation).

In Figure 5d we show how many pairs the TED* are exactly the same as TED. It is clear that more than 40% of the cases the two distances are exactly the same and for some cases the ratio can be up to 80%. This shows that TED* could be a good approximation to TED. Notice that, there exists low equivalency percentage for $k = 4/5$, because we can only compare the computable trees on both TED and TED* due to the hardness of TED and when $k = 4/5$, it is more likely to have "complex" structures where TED* has more "moving nodes in the same level" operations. When $k = 2/3$, the number of levels is too small to have "moving nodes in the same level" operations and when $k > 5$, the computable trees for TED are generally in simple structures and also few "moving nodes in the same level" operations can happen. That's why we can observe that for the computable trees with $k = 4/5$, the equivalency percentage is relatively low.



(a) TED* Computation Time (b) NED Computation Time

Figure 6: Computation Time of TED* and NED

7.2 TED* and NED Computation

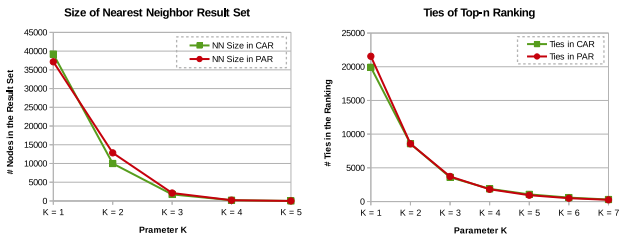
In Figure 6a, we plot the time to compute TED* with different tree sizes. In this experiment, we extract 3-adjacent trees from nodes in (AMZN) graph and (DBLP) graphs respectively. As shown in the Section 7.1, the exact TED and GED cannot deal with trees and graphs with more than 10 nodes. However, TED* is able to compute the distance between a pair of trees with up to 500 nodes in one millisecond.

Figure 6b plots the NED computation time for different tree levels as k changes. Firstly the time of k-adjacent tree retrieval is negligible compared to TED* computation. Thus, according to the experiments, there is no time difference between TED* and NED. Also along with increasing k, the computation time increases exponentially. Basically number of nodes per level increases exponentially and the performance is dominated by Hungarian algorithm. In the figure, when the value of k is under 5, the computation time is within one millisecond. Next, we show that, the parameter k does not need to be very large (5 is large enough) for nearest neighbor queries and top-n ranking queries to give meaningful results.

7.3 Analysis of Parameter k

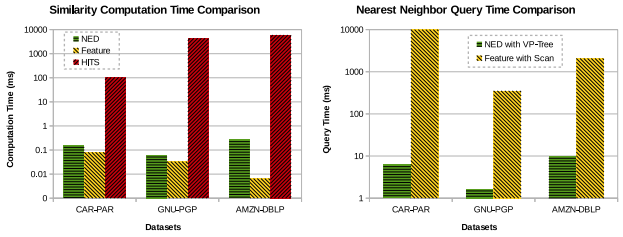
There is only one parameter k in NED which is number of levels to be considered in the comparison. In this section, we use nearest neighbor and top-n ranking queries to show the effects of parameter k on the query results.

The nearest neighbor query task is the following: for a given node in one graph, find the l most similar nodes in another graph. When the parameter k is small, more nodes in the graph can have the same minimal distance (usually 0) to a given node. When k increases, the NED increases monotonically as proved in Section 5.4. Therefore, by choosing different parameter k, we can control the number of nodes



(a) Nearest Neighbor Query (b) Top-n Ranking

Figure 7: Analysis of Parameter k in NED



(a) Computation Time (b) NN Query Time

Figure 8: Node Similarity Comparison

in the nearest neighbor result set. Figure 7a shows the number of nodes in the nearest neighbor result set for different values of k. In the experiment, we randomly pick 100 nodes from (CAR) and (PAR) graphs as queries and the nodes in the other graph are computed. It is obvious that when the parameter k increases, the number of nodes in the nearest neighbor result set decreases.

The effect of parameter k for the top-n ranking query indicates how many identical distances (ties) that appear in the ranking. As shown in Figure 7b, the ties start to break when k increases. Intuitively, it is more likely to have isomorphic neighborhood structures if fewer levels of structures are considered. Figure 7b shows the number of ties in the top-n ranking for different values of k. The experimental setting is the same as in the nearest neighbor query.

Choosing a proper value for the parameter k depends on the query speed and quality. When k increases, the computation time of NED increases as shown in Section 7.2. On the other hand, when k increases, both the number of nodes in the nearest neighbor result set and the number of ties in the ranking decreases. So it is clear that there exists a trade-off between the query speed and quality. Furthermore, the proper value of k depends on the specific applications that the graphs come from. For example, the denser graphs with nodes having larger average degrees usually need smaller k.

7.4 Query Comparison

In this section, we compare the performance of NED with other existing inter-graph node similarity measures: HITS-based similarity and Feature-based similarity.

Figure 8a shows the distance computation time for NED, HITS-based similarity, and Feature-based similarity. In this experiment, we extract 5-adjacent trees for the nodes in (CAR) and (PAR) graphs and 3-adjacent trees for the nodes in (PGP), (GNU), (AMZN) and (DBLP) graphs. NED, HITS-based similarity, and Feature-based similarity are computed over random pairs of nodes and the average computation time for different measures are shown in Figure 8a.

From this figure, it is clear that HITS-based similarity is the slowest among all three methods, because the HITS-based similarity iteratively updates the similarity matrix until the matrix converges. Feature-based similarity is faster than NED which makes sense since Feature-based similarity only collects statistical information from the neighborhood. NED pays a little extra overhead to take into account more topological information and be a metric. We show later why more topological information and metric matter.

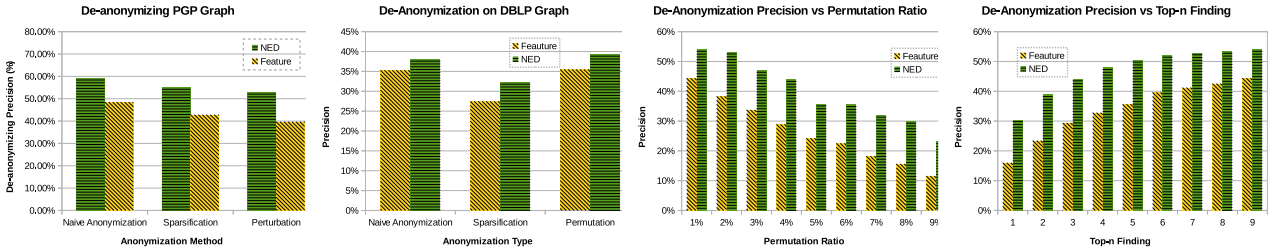
As discussed, the Feature-based similarity discards certain topological information which makes it not precise. We use graph de-anonymization in Section 7.5 to show that, with more topological information, NED can achieve a higher precision in de-anonymization compared to the Feature-based similarity since NED captures more topological information.

Also since the Feature-based similarity uses different features for different pairs, the similarity values of two pairs of nodes are not comparable. When using the Feature-based similarity for nearest neighbor queries, a full scan is necessary. However, as a metric, NED has the advantage in being used with existing metric indices for efficient query processing. Figure 8b shows that although NED pays a little bit more time than Feature-based similarity in distance computation, by combining with a metric index (existing implementation of the VP-Tree), NED is able to execute a nearest neighbor query much faster (orders of magnitude) than the Feature-based similarity.

7.5 Case Study: De-anonymizing Graphs

In this section, we use graph de-anonymization as a case study to show the merits of NED. Graph de-anonymization is a popular privacy topic in database and data mining [22]. When publishing data, the data owners tend to anonymize the original data firstly by using anonymization techniques such as naive anonymization, sparsification, perturbation and so on. However, there may be certain information inferred by using domain knowledge. Using small portion of exposed information to de-anonymize the anonymous graph is a typical privacy risk in database and data mining. Also the ease of de-anonymization can be used to check the quality of anonymization. Ji et al. [9] provides an extensive survey on graph de-anonymization which analyze the state-of-art techniques comprehensively. Basically for the seed-free and auxiliary graph-free approaches, the feature-based similarity is the major technique to adopt.

In this experiment, we simulate the de-anonymization process by splitting (PGP) and (DBLP) graphs into two parts: training data and testing data. The training data is the subgraph with identification, while the testing data is the anonymous subgraph with naive anonymization, sparsification, and perturbation. For each node in the anonymous graph, we try to assign the identity by finding top similar nodes in the training data (guess-based approach). Since NED captures more topological information, it is capable to detect more sophisticated differences between structures. Therefore, NED can achieve much higher precision than the feature-based similarity. Note that HITS similarity has limitations on performance which prevent it to be a feasible solution on de-anonymization.



(a) De-Anonymize PGP (b) De-Anonymize DBLP (c) Permutation Ratio (d) Top-n Finding

Figure 9: Graph De-Anonymization

In Figure 9a and Figure 9b, we show the precision of de-anonymization using NED and feature-based similarity. In the experiment, the parameter k is set to 3 and we examine the top-5 similar nodes (best 5 matches) in (PGP) data and the top-10 similar nodes in (DBLP) data. The permutation ratio in (PGP) is 1% and in (DBLP) is 5%. Based on the results, NED is able to identify anonymized nodes with better accuracy than the feature-based similarity.

In Figure 9c, we show how the precision changes by varying permutation ratio. The precision of NED reduces slower than feature-based similarity when the permutation ratio increases. The Figure 9d shows that when more nodes in top-n results are examined, how the de-anonymization precision changes. It is clear that if fewer nodes are checked which means there are less nodes in the top-n results, NED can still achieve a higher precision.

8. RELATED WORK

One major type of node similarity measure is called link-based similarity or transitivity-based similarity and is designed to compare intra-graph nodes. SimRank [8] and a number of SimRank variants like SimRank* [30], SimRank++ [2], RoleSim [10], just to name a few, are typical link-based similarities which have been studied extensively. Other link-based similarities include random walks with restart [27], influence-based similarity [13], and path-based similarity [25]. A comparative study for link-based node similarities can be found in [16]. Unfortunately, those link-based node similarities are not suitable for comparing inter-graph nodes since these nodes are not connected and the distances will be always 0.

To compare inter-graph nodes, neighborhood-based similarities have been used. Some primitive methods directly compare the ego-nets (direct neighbors) of two nodes using Jaccard coefficient, Sørensen–Dice coefficient, or Ochiai coefficient [29]. Ness [11] and NeMa [12] expand on this idea and they use the structure of the k-hop neighborhood for each node. However, for all these methods, if two nodes do not share common neighbors (or neighbors with the same labels), the distance will always be 0. Similarly, graph kernel-based methods [24, 23] also rely on the labels or attributes to align the nodes when using Weisfeiler-Lehman test.

An approach that can work for inter-graph nodes is to extract features from each node using the neighborhood structure and compare these features. OddBall [1] and NetSimile [3] construct the feature vectors by using the ego-nets (direct neighbors) information such as the degree of the node, the number of edges in the ego-net and so on. ReFeX [7] is an improved framework to construct the structural features recursively which is the state-of-art feature-based similarity measurement. The main problem with this approach is that the choice of features is ad-hoc and the distance values are hard to interpret. Furthermore, for the more advanced method, ReFeX, the distance function is not a metric due to the pruning process.

Another method that has been used for comparing biological networks, such as protein-protein interaction networks (PPI) and metabolic networks, is to extract a feature vector using graphlets [17, 6]. Graphlets are small connected non-isomorphic induced subgraphs of a large network [20]. However, these methods require expert knowledge for the pruning, otherwise are limited to extremely small neighborhood around each node.

Another node similarity for inter-graph nodes based only on the network structure is proposed by Blondel et al. [4] which is called HITS-based similarity. In HITS-based similarity, all pairs of nodes between two graphs are virtually connected. The similarity between a pair of inter-graph nodes is calculated using the following similarity matrix:

$$S_{k+1} = BS_kA^T + B^TS_kA$$

where A and B are the adjacency matrices of graphs and S_k is the similarity matrix in the k iteration.

Both HITS-based and Feature-based similarities are capable to compare inter-graph nodes without any additional assumption. However, HITS-based similarity is neither metric nor efficient. On the other hand, Feature-based similarities use ad-hoc statistical information which cannot distinguish minor topological differences. This means that Feature-based similarities may treat two nodes as equivalent even though they have different neighborhood structures.

9. CONCLUSION

In this paper, we study the inter-graph node similarity problem. A major application of inter-graph node similarity is transfer learning on graphs, i.e., learning a new graph based on the existing knowledge from another one. To address this problem, this paper proposes a novel distance function called NED. In NED, the k-adjacent trees between two nodes to be compared are firstly extracted. Then, a newly proposed modified tree edit distance called TED* is used to calculate the distance between two k-adjacent trees. TED* is a generic distance function for comparing trees which is easy to compute in polynomial time and satisfies the metric properties of the edit distance. Therefore, NED is a node metric. Due to the metric properties, NED

is compatible with existing metric indexing methods. Moreover, since NED captures more structural information, it is demonstrated to be more effective and precise for graph de-anonymization. NED, as an inter-graph node similarity, can be used to compare two nodes in two different graphs with totally different labeling systems (or even without any label at all), since NED is a pure neighborhood-topology-based metric.

10. ACKNOWLEDGMENTS

This work was partially supported by NSF grants IIS-1320542 and CNS-1414119.

11. REFERENCES

[1] L. Akoglu, M. McGlohon, and C. Faloutsos. oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421, 2010.

[2] I. Antonellis, H. Garcia-Molina, and C. Chang. Simrank++: query rewriting through link analysis of the click graph. *PVLDB*, 1(1):408–421, 2008.

[3] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos. Network similarity via multiple social theories. In *ASONAM*, pages 1439–1440, 2013.

[4] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4):647–666, 2004.

[5] C. Clark and J. Kalita. A comparison of algorithms for the pairwise alignment of biological networks. *Bioinformatics*, 30(16):2351–2359, 2014.

[6] D. Davis, Ö. N. Yaveroglu, N. Malod-Dognin, A. Stojmirovic, and N. Przulj. Topology-function conservation in protein-protein interaction networks. *Bioinformatics*, 31(10):1632–1639, 2015.

[7] K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: graph mining using recursive structural features. In *KDD*, pages 663–671, 2011.

[8] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[9] S. Ji, P. Mittal, and R. Beyah. Graph data anonymization, de-anonymization attacks, and de-anonymizability quantification: A survey. *IEEE Communications Surveys Tutorials*, PP(99), 2016.

[10] R. Jin, V. E. Lee, and H. Hong. Axiomatic ranking of network role similarity. In *KDD*, pages 922–930, 2011.

[11] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD*, pages 901–912, 2011.

[12] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. *PVLDB*, 6(3):181–192, 2013.

[13] D. Koutra, N. Shah, J. T. Vogelstein, B. Gallagher, and C. Faloutsos. Deltacon: Principled massive-graph similarity function with attribution. *TKDD*, 10(3):28, 2016.

[14] J. Kunegis. KONECT: the koblenz network collection. In *WWW*, pages 1343–1350, 2013.

[15] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[16] H. Liu, J. He, D. Zhu, C. X. Ling, and X. Du. Measuring similarity based on link information: A comparative study. *IEEE Trans. Knowl. Data Eng.*, 25(12):2823–2840, 2013.

[17] N. Malod-Dognin and N. Przulj. L-GRAAL: lagrangian graphlet-based network aligner. *Bioinformatics*, 31(13):2182–2189, 2015.

[18] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *SSPR*, pages 163–172, 2006.

[19] M. Pawlik and N. Augsten. Rted: A robust algorithm for the tree edit distance. *PVLDB*, 5(4):334–345, 2011.

[20] N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric. *Bioinformatics*, 20(18):3508–3515, 2004.

[21] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comp.*, 27(7):950–959, 2009.

[22] K. Sharad and G. Danezis. An automated social graph de-anonymization technique. In *WPES*, pages 47–58, 2014.

[23] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[24] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009.

[25] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4(11):992–1003, 2011.

[26] K. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[27] H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[28] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *IEEE Trans. Knowl. Data Eng.*, 24(3):440–451, 2012.

[29] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: a structural clustering algorithm for networks. In *KDD*, page 8240833, 2007.

[30] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1):13–24, 2013.

[31] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[32] K. Zhang and T. Jiang. Some MAX snp-hard results concerning unordered labeled trees. *Inf. Process. Lett.*, 49(5):249–254, 1994.

[33] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3):133–139, 1992.

[34] H. Zhu, X. Meng, and G. Kollios. NED: An inter-graph node metric on edit distance. *CoRR*, abs/1602.02358, 2016.