GEDet: Adversarially Learned Few-shot Detection of Erroneous Nodes in Graphs

Sheng Guan* Peng Lin† Hanchao Ma* Yinghui Wu*
*Case Western Reserve University Washington State University
{sxg967,hxm382,yxw1650}@case.edu peng.lin@wsu.edu

***Abstract*—Detecting nodes with erroneous information in graphs is important yet challenging, due to the lack of examples and the diversified scenarios of errors. We introduce GEDet, a few-shot learning based framework to detect erroneous nodes in graphs. GEDet consists of two novel components, each addresses a unique challenge. (1) To cope with the lack of examples, we introduce a graph augmentation module to enrich training labels. The module not only generates additional synthetic training labels by simulating different erroneous scenarios, but also exploits non-local relations to enrich neighborhood information. (2) To further improve the accuracy, we introduce an adversarially learned module that can better detect erroneous nodes by distinguishing nodes with synthetic and real labels encoded by graph autoencoders. Unlike conventional error detection models, GEDet yields effective classifiers that are optimized for a few yet diversified examples in the presence of multiple error scenarios. We show that using only a small number of examples, GEDet significantly improves the competing methods such as constraint-based detection and anomaly detection, with a gain of 35% on recall, and 30% on precision.**

***Index Terms*—data cleaning, few-shot learning, adversarial deep learning, graphs**

I. Introduction

Ensuring high-quality graph data such as knowledge bases and social networks is important for various downstream applications. The cornerstone task is to detect erroneous nodes that have wrong values in attributed graphs. Various approaches have been studied to curate graph data [20]. Nevertheless, these methods focus on inferring new information from their correct counterparts and cannot be directly used to detect erroneous nodes that are already in graphs.

Detecting errors has been extensively studied for relational data [1], and has been recently extended to attributed graphs. These methods are optimized for "pre-emptively" assumed erroneous scenarios. Erroneous nodes with incorrect attribute values can be captured as violations of data constraints [10], anomalies [17], or via statistical inference [19]. These methods can be accurate for specific types of errors, yet may not achieve good performance in the presence of multiple types of errors. Consider the following example from real knowledge graphs.

Example 1: Fig. 1 illustrates a fraction of a knowledge graph about films. Each node carries a type (*e.g.,*film) and a set of attributes (*e.g., 'title'*) with values (e.g., "*HarryPotter_I*"). Each edge carries the relationships between nodes (e.g., screenWriterOf). There are three erroneous nodes[1]):

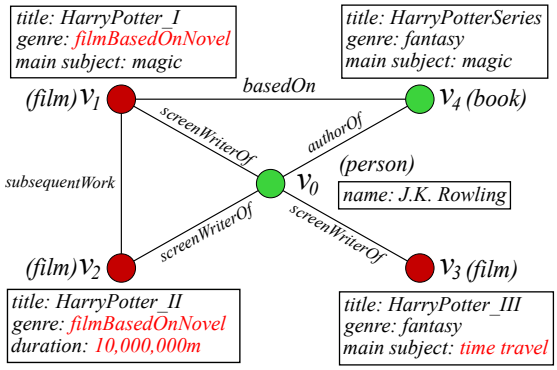


Fig. 1: Detecting Erroneous Nodes in Knowledge Graphs

- Film v_1 has a genre "filmBasedOnNovel" that should be "fantasy", same as the genre of the book v_4.
- Film v_2 has the same genre as v_1 that should be "fantasy", and a duration "10,000,000m" that should be "161m".
- Film v_3's main subject "time travel" should be "magic".

Multiple erroneous scenarios exist for a single type of nodes *e.g.,* "film". For example, (1) the erroneous node v_1 can be detected as its "genre" violates a graph data constraint φ [10] that states "if a movie is based on a book ('*basedOn*'), then they should have the same genre and main subject"; (2) node v_2 has an outlier over the values of the *duration* of films.

Nonetheless, applying individual methods or a simple "union" of their results may not capture all the errors from different scenarios. For example, node v_3 may not be captured by outlier detection as "time travel" can be a common main subject. The constraint φ also fails to capture v_3, as no direct link connects v_3 and v_4. The union of the results from the two approaches overlapped at v_2, and v_3 remains undetected. □

We identify the following challenges.

Multiple Erroneous Scenarios. Erroneous nodes are characterized by more than one type of errors (misspelling, violation of value constraints, numerical outliers) [28], [20]. Such a diversity cannot be fully captured by a single model. For example, data constraints and outlier detection may achieve good precision but often a low recall; a simple combination does not ensure coverage due to overlapped errors [1].

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

[1]https://www.wikidata.org/w/index.php?title=Q102438&action=history

Lack of Examples. It is often difficult to obtain a full set of correctly labeled examples, especially over real graphs. This issue remains in error detection by crowd-sourcing [5], due to labels with low-quality. This limits the capability of learning-based approaches that detect errors via node classification.

Another source of sparsity comes from the inherent incompleteness of real-world graphs [20]. Specifically, missing edges leads to the insufficient neighborhood, hence may downgrade the performance of error detection (*e.g.,* constraint-based detection, or message-passing based learning [31]).

The above challenges call for graph error detection approaches that can utilize a few examples from different error scenarios to achieve good performance.

Contribution. This paper proposes Graph Error Detection (GEDet), a few-shot learning-based framework to identify erroneous nodes in graphs. GEDet addresses the above challenges with the following novel features.

Few-shot Error Detection. GEDet enables few-shot learning [30] based error detection framework with a novel *graph augmentation* strategy. It enhances supervised information with both synthetic labels and non-local neighbors.

(1) GEDet generates synthetic erroneous examples with an error generative model. The error generation combines transformations to simulate various types of errors characterized including data anomalies and graph data constraints.

(2) Moreover, GEDet actively identifies useful nodes with features that may contribute to error detection via label propagation. It enhances node neighbors via a biased link inference process that favors such non-local counterparts. This further mitigates the impact of sparse neighbors due to missing edges.

Adversarially Learned Detection. The detection module of GEDet exploits the principle of adversarial learning to improve the accuracy of error detection. In a nutshell, GEDet learns a generator to simulate the distribution of real labels from graph autoencoders over the augmented graphs to "fool" a discriminator. By forcing the discriminator to differentiate not only "error" and "correct" but also "synthetic" and "real" labels, GEDet improves the accuracy of error detection.

Achieving high recall with reasonable precision has been a difficult task for existing error detection [1]. The few-shot module of GEDet ensures its coverage of heterogeneous errors (hence improves the recall), and the adversarial model improves the decision boundary of classification (hence retains reasonable precision). This ensures GEDet a robust performance on error detection over graphs with multiple error scenarios, as verified by our experiments.

II. Preliminaries on Error Detection in Graphs

A. Error Detection Problem

Graphs. A graph $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq V \times V$. Each node $v \in V$ (resp. edge $e \in E$) carries a node type $L(v)$ from a set $\mathcal{T}$ (resp. relation $L(e)$ from a set $\mathcal{R}$). Each node $v \in V$ also carries a vector $(v.A_1, \ldots v.A_n)$ defined on n attributes, where $v.A_i = a_i$ ($i \in [1, n]$) denotes that the attribute A_i of v has value a_i.

Given a graph $G = (V, E)$, we consider a "ground truth" node v^* for each node $v \in V$ that carries correct values of all the attributes of v. A node v is *erroneous* (annotated by a class label 'error') if there exists an attribute A, such that $v.A \neq v^*.A$ ($v.A$ can be 'null' that denotes a missing value); otherwise, v is *correct* ($v = v^*$; annotated by a class label 'correct'). A node is *labeled* if its class label is known; otherwise it is *unlabeled*.

Problem statement. Error detection in a graph G=(V, E) can be formulated as a node classification problem. The *training nodes* $V_{\mathcal{T}} = V^e \cup V^c$ ($V_{\mathcal{T}} \subseteq V$) refers to a set of labeled nodes, where (a) V^e is a set of erroneous nodes, and for each node $v \in V^e$, a correct counterpart v^* is known; and (b) V^c is a set of correct nodes with labels 'correct'.

Given graph G and the training nodes $V_{\mathcal{T}}$, the error detection problem is to learn a model that accesses G to infer the class labels of a set of unlabeled *test nodes* from $V \setminus V_{\mathcal{T}}$.

B. Enabling Techniques

GEDet is empowered by several established techniques that are nontrivially enhanced to improve the accuracy of error detection in graphs. We next revisit these techniques.

Graph Representation Learning. GEDet is empowered by graph representation learning, which aims to derive proper vectorized representation of graphs that can be used for downstream tasks *e.g.,* classification.

Graph representation. Graph learning requires a proper encoding of attributed graphs as feature matrix. Given a graph G=(V, E), a feature representation of G is a pair (X_G, A_G), where (1) X_G is a node feature matrix ($X_G \in \mathbb{R}^{|V| \times d}$) that records for each node $v \in V$ a feature vector $x_v \in \mathbb{R}^d$; and (2) A_G is the adjacency matrix of G.

GEDet uses a library of established encoding functions (*e.g.,* word embedding [11]) to convert heterogeneous nodes to feature space $\mathbb{R}^d$. To encode different types of nodes it concatenates type as one-hot feature [23], and applies meta-path based propagation [33], which groups the neighbors of a node according to their similarity and distances following meta-path connections (see Section IV).

Graph Convolutional Autoencoders. GEDet exploits graph autoencoders (GAE) [31], [15] to learn node representations for downstream error detection. A graph autoencoder takes as input a graph representation $G = (X_G, A_G)$ to learn an encoding $Z \in \mathbb{R}^{|V| \times d'}$ ($d' \ll d$) of X_G (by an encoder Enc), from which reconstructing (X_G, A_G) is possible (by a decoder Dec). That is, the learning minimizes a reconstruction loss:

$$\min \mathrm{dist}((X_G, A_G), \mathrm{Dec}(\mathrm{Enc}(X_G, A_G)) \quad (1)$$

which is quantified by a distance metric dist.

A majority of successful applications of GAEs relied on graph convolutional networks (GCN) [14] to encode nodes due to its simplicity and linear complexity. The ℓ-th layer of a GCN takes a feature vector $h^{(\ell-1)}(\mathbf{x}_v)$ for each node v as

Symbol	Notation
$G = (V, E)$	Heterogeneous graph with nodes V and edges E
$V_{\mathcal{T}} = V^e \cup V^c$	Labeled training nodes; V^e: erroneous nodes; V^c: correct nodes
(X_G, A_G)	feature representation of G: (X_G: feature matrix; A_G: adjacency matrix)
$h^{(n)}(\mathbf{x}_v)$	the embedding of node v (at layer n)
$H^{(n)}$	graph embedding matrix at layer n
$\mathcal{H}$	the error generative model
$\mathcal{L}$	the link inference model
$\mathcal{Z}$	the graph convolutional autoencoder
$\mathcal{G}$, $\mathcal{D}$	generator and discriminator of GAN
$L(\mathcal{G})$, $L(\mathcal{D})$	loss function of $\mathcal{G}$ and $\mathcal{D}$
L^s, L^u	supervised and unsupervised loss of $L(\mathcal{D})$

TABLE I: Table of Notations

input ($h^0(\mathbf{x}_v)$ refers to node features $\mathbf{x}_v$). Node v updates its feature vector $h^\ell(\mathbf{x}_v)$ by aggregating the counterparts from its neighborhood $\mathcal{N}$(v). As updated feature vector $h^\ell(\mathbf{x}_v)$ becomes the input to the (ℓ + 1)-th layer, the aggregation propagates from layer 1 to n as follows:

$$H^{(\ell+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}\right) \quad (2)$$

where $\tilde{A} = A + I_{|V|}$ is the adjacency matrix of G with self-loops ($I_{|V|}$ refers to the identity matrix), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $W^{(\ell)}$ is a layer-specific trainable weight matrix, $\sigma(\cdot)$ is an activation function, and $H^{(\ell+1)}$ is the output at the ℓ^{th} layer.

GEDet specifies a GAE as follows. (1) The general term Enc is specified with a n-layer GCN (n=2 by default). (2) The decoder consists the following. (a) A structure decoder reconstructs the original network structure with the learned representations Z ($\hat{A}_G = \sigma(ZZ^{\mathrm{T}})$). (b) A feature decoder reconstructs X_G ($\hat{X}_G = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}ZW^{(2)})$).

Few-shot Learning. Few-shot learning [30] aims to learn effective models from a few labeled examples, by *e.g.,* enhancing augmented training data. This can be achieved by using (handcrafted) rules [22]. GEDet capitalizes on transformations to simulate error generation and produce a set of nodes with synthetic erroneous attribute values (with a label "synthetic error"). It also consistently enhances neighbors of nodes to better improve accuracy (see Section IV).

Generative Adversarial Networks. Generative adversarial networks (GANs) are used to generate synthetic data that approximates real data distribution by jointly learning a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. The generator $\mathcal{G}$ produces synthetic data as close to the real counterpart as possible, and $\mathcal{D}$ learns to distinguish real embeddings from synthetic counterparts from $\mathcal{G}$. It has been shown that GANs can improve semi-supervised learning [25], [6] with improved decision boundary when the target model is forced to distinguish true and synthetic labels.

C. Error Detection as "Two-player Game"

GEDet models error detection as a two-players game between a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. It exploits the principle of adversarial detection with the following intuition.

- A generator $\mathcal{G}$ tries its best to "fool" a discriminator $\mathcal{D}$ by representing the augmented G (including synthetically labeled nodes) as close to the real counterpart as possible.
- The discriminator $\mathcal{D}$ meanwhile learns to distinguish nodes with real labels from the candidates from $\mathcal{G}$.

The "competition" between $\mathcal{G}$ and $\mathcal{D}$ in a joint learning process iteratively improves the performance of both: $\mathcal{G}$ eventually learns the distribution of the real labels, and $\mathcal{D}$ improves accuracy of error detection.

Consider the labeled set $\mathbb{L} = \{(x, y)\}$, where x represents a node and $y \in \{$'$error$'$(y = 1)$, '$correct$' $(y = 2)\}$. We introduce a third label 'synthetic error' ($y = 3$) to annotate an embedding of a node with synthetic errors from $\mathcal{G}$. We can enforce a learning objective of $\mathcal{D}$ in the form of:

$$\max_{\mathcal{D}} \mathbb{E}_{x\sim\mathbb{L}} \log P_{\mathcal{D}}(y|x, y \le 2) + \mathbb{E}_{x\sim p} \log P_{\mathcal{D}}(y \le 2|x) + \mathbb{E}_{x\sim p_{\mathcal{G}}} \log P_{\mathcal{D}}(3|x) \quad (3)$$

where p and $p_{\mathcal{G}}$ refers to the distribution of the real data and its synthetic counterpart from $\mathcal{G}$, respectively. The first, second, and third terms are to maximize the log conditional probability for (1) the labeled nodes with true labels, (2) the unlabeled nodes (as either 'error' or 'correct'), and (3) the generated synthetic examples (into the class of 'synthetic error').

Although desirable, GANs cannot be directly applied for error detection in graphs. GEDet obtains a compact encoding (including error distribution and augmented topology of original graphs) from graph autoencoders. The generator and discriminator then play a competitive game over the compact encoding to improve the decision boundary by specifying and optimizing the above objective (Section V).

III. Framework Overview

A. GEDet Models

We start with the cornerstone models in GEDet framework.

Error Generative Model ($\mathcal{H}$). We advocate to learning real node error distribution at the attribute level, and characterize various error scenarios with *transformations*. A transformation simulates error generation process by "perturbing" a correct attributed value to a mismatched counterpart. Given labeled nodes $V_{\mathcal{T}}$ and a library of transformations Ψ, the error generative model $\mathcal{H}$ converts the node attribute values of $V_{\mathcal{T}}$ to approximate real error distribution with observed conditional distribution of the node attribute values.

Link Inference Model ($\mathcal{L}$). To mitigate the impact of missing neighborhood, GEDet adopts a link inference model $\mathcal{L}$ to infer links to non-local nodes that may contribute to the local representation learning of a node via layer-wise propagation.

GEDet applies the error generative mode $\mathcal{H}$ and link inference model $\mathcal{L}$ to perform graph augmentation, with synthetic labels and enriched neighbors, respectively (see Section IV).

Error Detection Model ($\mathcal{Z}$, $\mathcal{D}$ and $\mathcal{G}$). GEDet detects errors with a graph convolutional autoencoder model $\mathcal{Z}$ and a GAN model. Graph autoencoder $\mathcal{Z}$ generates the robust graph embedding as input for GAN. GAN consists (1) a generator $\mathcal{G}$ to produce a synthetic graph sample with labels from training nodes and synthetic ones, and (2) a discriminator $\mathcal{D}$ that transforms an input graph sample to scalar values, and further infers node class labels and distinguishes real and synthetic labels for unlabeled nodes via a classification layer.

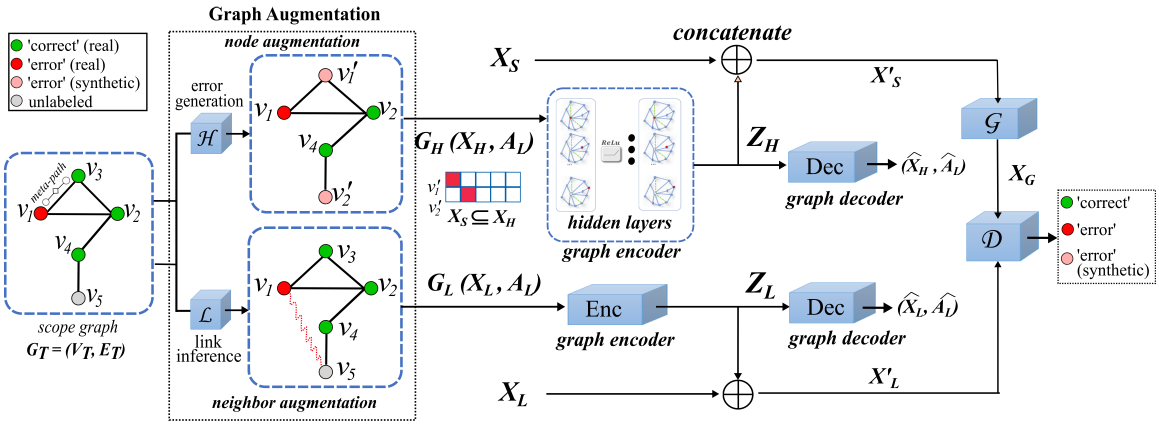


Fig. 2: Overview of GEDet Framework

B. GEDet Framework

We next provide an overview of GEDet framework (illustrated in Fig. 2). GEDet takes as input with an attributed graph $G = (V, E)$, a set of training nodes $V_{\mathcal{T}} \subseteq V$ with labels, and a library of transformation functions Ψ (Section IV-A).

Scope Generation. This module induces from the original graph G a *scope* G_T to facilitate graph learning. This step necessarily converts attributed input G to uniform feature representation. Specifically, for each node type $l \in \mathcal{T}$, GEDet induces a corresponding scope graph $G_l = (V_l, E_l)$, where

- $V_l \subseteq V$, $L(v) = l\ \forall v \in V_l$;
- $(v, v') \in E_l \cap R$.

where $E_l \subseteq V_l \times V_l$, and R is a node proximity relation induced by a set of meta-paths [26]. A meta-path Φ is a path in the form of $\mathcal{T}_1 \xrightarrow{R_1} \mathcal{T}_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} \mathcal{T}_{l+1}$ (abbreviated as $\mathcal{T}_1\mathcal{T}_2 \dots \mathcal{T}_{l+1}$), which describes a composite relation $R = R_1 \circ R_2 \circ \dots \circ R_l$ between entities $\mathcal{T}_1$ and $\mathcal{T}_{l+1}$, where $\circ$ denotes the composition operator on relations. Following [26], GEDet asserts that two nodes (v, v') of the same type are similar ($(v, v') \in R$) if they are connected by a symmetric meta path with a high node proximity score.

Given a set of targeted node type $T' \subseteq \mathcal{T}$ induced from *e.g.,* test node set, a scope graph $G_T = (V_T, E_T)$ is induced as the union of each type-specific scope graph $G_l = (V_l, E_l)$ for each $l \in T'$, *i.e.,* $V_T = \bigcup_{l \in T'} V_l$, $E_T = \bigcup_{l \in T'} E_l$. The feature representation $G_T = (X_T, A_T)$ is then derived as the input for graph augmentation.

Graph Augmentation. This module augments the input scope graph G_T by the following procedures.

Node augmentation. GEDet learns the error generative model $\mathcal{H}$ from the training nodes V_T following the data distribution in G_T and exploits transformations Ψ. $\mathcal{H}$ is used to generate a set of nodes with transformed erroneous values and a synthetic "error" label. This yields a graph representation G_H with node features X_H, which includes a set of synthetic feature vectors X_S obtained from the nodes with transformed attribute values.

Neighbor augmentation. Based on the link inference model $\mathcal{L}$, GEDet further enhances the neighbors of the nodes in the scope graph G_T by inferring non-local relationships that favor error detection. This introduces a set of "virtual neighbors" for the nodes in G_T, and transforms the adjacency matrix A_T of G_T to an enriched counterpart A_L.

The graph augmentation (illustrated in Fig. 2) yields the following representations: (1) $G_H = (X_H, A_L)$ with features encoding transformed attribute values; and (2) $G_L = (X_T, A_L)$, with the original features X_T from the scope G_T. Both G_H and G_L share the augmented topology A_L.

Graph augmentation is quite effective: using only 10 examples, it improves the accuracy of error detection with a gain of 10%, as verified in Section VI.

Adversarial Detection. This module learns compact node embeddings with graph autoencoder $\mathcal{Z}$ and generates classifiers (as illustrated in Fig. 2). (1) The graph autoencoder $\mathcal{Z}$ derives the latent representation Z_H and Z_L of G_H and G_L, respectively. The encoding Z_S of synthetic nodes X_S can be induced via node index. (2) The input $X'_S = Z_S \oplus X_S$ for the generator $\mathcal{G}$ is computed by concatenating Z_S and X_S; the input $X'_L = X_L \oplus Z_L$ for discriminator $\mathcal{D}$ is computed similarly. GEDet then jointly learns $\mathcal{G}$ and $\mathcal{D}$ in the competitive game, and derives the class labels at the prediction stage.

The adversarial learning further improves the accuracy of error detection with a gain of 11% compared with GEDet-B, as verified in our experimental study.

We next introduce the details of graph augmentation and error detection modules, in Sections IV and V, respectively.

IV. Graph Augmentation

A. Augmentation of Labeled Nodes

The node augmentation module learns an error generative model $\mathcal{H}$ to approximate error distribution of the scope graph G_T, and produces synthetic erroneous nodes.

Characterizing Heterogeneous Errors. GEDet characterizes various erroneous scenarios with *transformation functions* (or simply transformations). Given a correct node v^* with attribute $v^*.A$, a transformation $\phi \in \Psi$ converts $v^*.A = a^*$ to a mismatched counterpart a ($\phi(v^*.A) \longmapsto a(a \neq a*)$), thus leads to an erroneous counterpart.

A transformation can be derived and applied to multiple nodes and multiple attributes, and can be specified in various forms such as (string) transformation rules [12], numerical errors [4], or induced from violations of data constraints [10].

Example 2: A semantic constraint φ' defined on films that states "a film m and a book b should have the same genre if m is based on b (Example 1)" can be expressed as a rule basedOn(m, b) $\rightarrow$ m.*genre* = b.*genre*. A corresponding transformation rule can be derived to "violate" φ': if node v_m (resp. v_b) in G has type *film* (resp. *book*) and basedOn(v_m, v_b), then change v_m.*genre* such that v_m.*genre* $\neq v_b$.*genre*. □

Weighted transformations. An erroneous node may be caused by multiple transformations. Given a library of transformations Ψ, and a set of erroneous nodes $V^e \subseteq V_{\mathcal{T}}$ (with the corresponding "ground truth" set V^*), the error generative process aims to simulate the transformations from each ground truth v^* to corresponding $v \in V^e$ as close as possible.

Given that v^* and v *should* refer to *same* real-world entity, we introduce a closeness measure between v with n attributes and a weighted transformation of its ground truth v^* as

$$F(v, \Psi(v^*)) = \sum_{i}^{n} \sum_{\phi_j \in \Psi} w_j \cdot \mathsf{sim}(v.A_i, \phi_j(v^*.A_i)) \quad (4)$$

which computes the accumulated similarity (sim) over all attribute values and their transformed counterparts, and over all transformations in Ψ. If a transformation ϕ is not applicable to $v^*.A$, $\phi(v^*.A) = v^*.A$ by default.

The overall closeness is measured by the probability:

$$P(\Psi(V^*)|V^e) = \frac{1}{Z} exp(\Sigma_{v \in V^e} F(v, \Psi(v^*))) \quad (5)$$

where Z is a normalizer such that $P(\Psi(V^*)|V^e) \in [0, 1]$.

GEDet then learns the weights $W = \{w_1, \ldots w_m\}$ of the m transformations in Ψ from the training examples V^e and V^* such that the log-likelihood of their relevancy is maximized:

$$W^* = \arg\max_{W} \log P(\Psi(V^*)|V^e) \quad (6)$$

The above objective function can be learned by L-BFGS [27]. The overall learning cost is $O(c(|V^e||n||\Psi|)^2)$, with c the number of gradient computations performed by the optimization. Our experiments verify that the learning cost is affordable (see Section VI), and n (number of node attributes), Ψ, and $|V^e|$ are all small in practice.

Where to inject "errors"?. A potential impact from synthetic label injection is the skewed label distribution of neighbors, especially for message-passing based graph learning. When most neighbors of a node have a majority label, there is a higher chance for the node to be assigned a wrong label [7].

In response, GEDet uses a prioritization strategy. Denote the number of transformations in Ψ that are applicable to node $v \in V$ as $\psi(v)$. A ranking score of node v is computed as

$$r(v) = \begin{cases} \frac{\psi(v)}{|\Psi|} \cdot \frac{|N(v) \cap V^e|}{|V^e|} & \text{if } v \in V^c \text{ and } N(v) \cap V^e \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

where $N(v)$ is the neighbors of v in G_T. We sample top-k nodes in the graph (k=$\frac{|V^c|-|V^e|}{2}$ by default, to balance the number of labeled correct and erroneous nodes). Intuitively, it favors the correct nodes that have more erroneous neighbors and are more "vulnerable" to given transformations (as more attributes can be transformed). This "breaks" the links between correct nodes and erroneous ones and mitigate the impact from skewed neighbors as the correct nodes are transformed to create synthetic "erroneous" ones. If no such nodes can be found, it randomly samples k correct or unlabeled nodes.

Remarks. GEDet supports the following: (1) Assigning higher weights to transformations that violate high-quality data constraints for graphs [10], [16]; (2) User-defined transformations such as anomaly detection; and (3) (User-defined) composite transformations (*e.g.,* ϕ=$\phi_1\phi_2$) to simulate error generation process as a sequence of transformations. On the other hand, our case analysis verified that GEDet is less sensitive to the quality of data constraints compared with constraint-based error detection (see "Case study" in Section VI).

B. Neighborhood Augmentation

The neighborhood augmentation further enhances sparse neighbors to mitigate the impact of missing links. While link prediction has been well studied, inferring all the missing links with these methods is neither affordable nor necessary.

Our idea is to learn and apply a *biased* link inference model $\mathcal{L}$ that optimizes the likelihood of connecting a node to its (non-neighbor) counterparts that can "contribute" to improve error detection, and performs only necessary amount of link inference for efficient augmentation. To this end, we specialize supervised random walk [3] to the following strategy.

(1) GEDet samples a set S of "seed" nodes from $V_{\mathcal{T}}$ with heuristic node importance ranking measures, such as degree, betweenness or PageRank. For each node $v \in S$, it assigns a set of positive nodes $S(v)^+$ that (a) have the same class ('error' or 'correct'), and (b) also have a node similarity with v no smaller than a threshold θ_v, where θ_v is the smallest node similarity between v and its direct neighbors.

GEDet supports a built-in library of node similarity measures *e.g.,* Euclidean distance and node embedding similarity.

(2) GEDet then specializes a supervised random walk [3] to learn $\mathcal{L}$ as a function f_ω (parameterized by a learnable vector ω) that properly assigns strength values to each edge (v, v') in G_T. Our goal is to ensure a random walk, if following the probability induced by edge strengths (a "transit probability") from a seed v, is more likely to visit the nodes in $S(v)^+$:

$$\begin{aligned} &\min_{\omega} \mathcal{F}(\omega) = ||\omega||^2 \ s.t. \\ &\forall v \in S, \forall v' \notin S(v)^+, v'' \in S(v)^+ : p_{vv'} < p_{vv''} \end{aligned} \quad (8)$$

Here p is the vector of PageRank scores that depends on edge strengths, and $p_{vv'}$ refers to the PageRank score of node v' with v the selected source node.

Once the weighted adjacency matrix is learned, for each labeled node v in G_T, GEDet selects top k non-local neighbors with the same label and with the highest transit probability.

V. Adversarially Learned Error Detection

One may directly apply graph representation learning *e.g.,*GCN and generate a node classifier. Nevertheless, a node may still not be correctly detected due to label propagation from both real and synthetic ones, and from a skewed label distribution. GEDet improves the accuracy by capitalizing graph autoencoder and GAN, with an observation that is consistent with [7].

We introduce the learning objectives of $\mathcal{G}$ and $\mathcal{D}$ in the two-player game, and then present the learning algorithm.

Loss Function of Generator. The learning objective of the generator $\mathcal{G}$ is to minimize the difference between the synthetic graph embeddings X'_S and the real counterpart from X'_L. We define the loss function of $\mathcal{G}$ (denoted as $L(\mathcal{G})$) as the *feature matching loss* [25]:

$$L(\mathcal{G}) = \left\| Avg_{\mathbf{x}_v \in X'_L} \left(h^{(\ell)}(\mathbf{x}_v) \right) - Avg_{\mathbf{x}'_v \in X'_S} \left(h^{(\ell)}(\mathbf{x}'_v) \right) \right\|_2^2 \quad (9)$$

where $Avg(\cdot)$ computes the mean vector representation of a graph feature matrix. Correspondingly, $Avg_{\mathbf{x}_v \in X'_L} \left(h^{(\ell)}(\mathbf{x}_v) \right)$ represents the mean vector representation of all feature vectors of nodes in graph embedding X'_L; $Avg_{\mathbf{x}'_v \in X'_S} \left(h^{(\ell)}(\mathbf{x}'_v) \right)$ represents the mean vector representation of all feature vectors of nodes in synthetic graph embedding X'_S. Intuitively, $\mathcal{G}$ has higher chance to fool the discriminator $\mathcal{D}$ by faking node embeddings closer to the real counterpart by minimizing $L(\mathcal{G})$.

Loss Function of Discriminator. To enable discriminator $\mathcal{D}$ to recognize erroneous nodes, and also to differentiate nodes having real and synthetic labels, we quantify the loss of $\mathcal{D}$ as

- *Supervised loss* (L^s), which quantifies the loss of accuracy on node label classification ('error' or 'correct'); and
- *Unsupervised loss* (L^u), its counterpart on distinguishing synthetic or real labels.

The learning objectives for $\mathcal{D}$ is thus to minimize the bi-criteria loss $L(\mathcal{D})$, which is defined as

$$L(\mathcal{D}) = L^s + \lambda L^u \quad (10)$$

where the supervised loss L^s and the unsupervised loss L^u (balanced by a hyper-parameter λ) are defined as follows.

Supervised Loss. We use *softmax cross-entropy* to calculate the cross-entropy errors given the true labels in the real training graph. We denote the node embedding matrix generated by $\mathcal{D}$ as $H^{(\mathrm{n})}$, where $H^{(\mathrm{n})} \in \mathbb{R}^{N' \times 2}$ (N' is the number of nodes in X'_L). The supervised loss is defined as

$$L^s = -\frac{1}{|V_T|} \sum_{v \in V_T} \left[y_v \log(p_v) + (1 - y_v) \log(1 - p_v) \right] \quad (11)$$

where V_T is the labeled node set in G_L, y_v is the ground truth label of node v (1 for "error" and 0 for "correct"), and p_v refers to the predicted probability of v being "error", which can be computed by applying a softmax function on $H^{(\mathrm{n})}$.

Unsupervised Loss. The discriminator $\mathcal{D}$ also aims to classify the real or synthetic examples as accurately as possible by minimizing the unsupervised loss L^u. We define L^u as:

$$\mathrm{L}^u = -\mathbb{E}_{x_v \sim X'_L}[\log \mathcal{D}(x_v)] - \mathbb{E}_{x_v \sim X'_S}[\log(1 - \mathcal{D}(x_v))] \quad (12)$$

Algorithm: Adversarial Learning of GEDet

Input: real graph embedding X'_L, synthetic graph embedding X'_S, learning rate α

repeat

- Draw all synthetic nodes from synthetic graph embedding X'_S
- Draw all real nodes from real graph embedding X'_L
- Update the discriminator by descending gradients of losses: $\nabla_{\theta_\mathcal{D}} L(\mathcal{D})$
- Draw all synthetic nodes from synthetic graph embedding X'_S
- Draw all real nodes from real graph embedding X'_L
- Update the generator by descending gradients of losses: $\nabla_{\theta_\mathcal{G}} L(\mathcal{G})$
- Reduce learning rate α

until Convergence or reaching the predefined total iterations

---

Fig. 3: The Adversarial Learning Algorithm

where $\mathcal{D}(x_v)$ represents the probability that node v comes from the real data X'_L instead of synthetic data X'_S. Both X'_L and X'_S are fed to learn $\mathcal{D}$ by updating trainable variables.

Adversarial Learning. Following [24], $\mathcal{G}$ feeds X'_S to a sequence of transpose convolutions and batch normalization operations. $\mathcal{D}$ takes real feature matrix X'_L and its synthetic counterpart X_G that is generated from $\mathcal{G}$ as inputs. It also applies a stack of convolutions with batch normalization operations. To prevent overfitting, $\mathcal{D}$ applies regularization through dropout and global average pooling operations. To determine the probability of an input being real,$\mathcal{D}$ feeds the logits through the LogSumExp operation. It feeds the logits through the softmax function to obtain classification probability.

As illustrated in Fig. 3, GEDet trains generator $\mathcal{G}$ and discriminator $\mathcal{D}$ by iteratively minimizing their corresponding losses. In the main training loop, we use equation (9) and equation (10) to compute the discriminator loss $L(\mathcal{D})$ and generator loss $L(\mathcal{G})$. We optimize the gradients by applying Adam [13]) and update corresponding discriminator $\mathcal{D}$ and generator $\mathcal{G}$. At the end of each iteration, we reduce the learning rate (*e.g.,* multiplied by a decreasing ratio) to prevent an unstable training process due to high learning rate.

Error Prediction. Following Equation (11), a node embedding matrix $H^{(\mathrm{n})}$ from the real input X'_L is generated in $\mathcal{D}$, where $H^{(\mathrm{n})} \in \mathbb{R}^{N' \times 2}$. For each node in X'_L with a 2-dimensional vector of logits, after applying the softmax function, we can get the predicted class probabilities and choose the larger one as the predicted class ('error' or 'correct').

Analysis. We next justify how our adversarial detection module improves the accuracy of error detection. Our analysis follows from that GAN can help semi-supervised classification by forming fine-grained decision boundary [32], [7].

(1) Our generator $\mathcal{G}$ follows weighted transformation to generate synthetically labeled nodes. This not only enhances supervised information for error detection, but also reduce the influence of labeled nodes that may interplay from different "correct" and "error" clusters (recall the prioritized strategy of graph augmentation in Section IV-A). [7] interprets the loss

Dataset	$\|V\|$	$\|E\|$	# node types	# edge types	avg. # attr
DBP	2.2M	7.4M	73	584	4
OAG	0.6M	1.7M	5	6	2
Yelp	1.5M	1.6M	42	20	5

TABLE II: Overview of Real-world Graphs

Dataset	$\|V_T\|$	$\|E_T\|$	avg. # attr	$\|V_{\mathcal{T}}\|$	$\|V^e\|$
Agent(DBP)	216	212	3	13	3
MusicGenre(DBP)	164	402	3	10	3
Transportation(DBP)	637	642	3	38	4
Species(DBP)	17.7K	20K	4	1062	134
Data Mining(OAG)	11.2K	12.9K	3	670	158
Machine Learning(OAG)	3.4K	3.3K	3	203	54
UserGroup1(Yelp)	3.4K	2.6K	3	202	57
UserGroup2(Yelp)	3.3K	2.5K	3	196	45

TABLE III: Scope Graphs (annotated by type)

$L(\mathcal{D})$ from the graph Laplacian regularization perspective:

$$L(\mathcal{D}) = L^s + \lambda \sum_{v_i, v_j \in V}^{i \neq j} \alpha_{ij} \cdot neq\left(y'_i, y'_j\right) \tag{13}$$

y'_i denotes predicted label of v_i. $\mathrm{neq}(\cdot,\cdot)$ is a 0-or-1 function representing not equal. α_{ij} is a regularization parameter that enables synthetic nodes to affect discriminator loss by changing the degree of node v_i, denoted as $\deg(i)$.

$$\alpha_{ij} = \frac{A_{ij}}{\sqrt{\deg(i) deg(j)}}, (i \neq j) \tag{14}$$

(2) As a first step, $\mathcal{G}$ generates synthetic samples in low-density areas that lie between error and correct node manifolds. In the learning process, discriminating synthetic samples from real nodes in $\mathcal{D}$ makes the classifier be aware of low-density areas, which benefits separating correct and error node clusters. The classifier tends to learn a higher curvature function and more accurately captures the boundary of different clusters. Inside each cluster, the confidence of classification boosts as the supervised loss L^s decreases and the decision boundary gets refined. With sufficient synthetic nodes lying between "correct" and "error" clusters, we link synthetic nodes to the nearest real nodes to regularize the graph Laplacian loss.

VI. Experiment

Using real-world graphs, we experimentally verify the effectiveness and the training cost of GEDet framework.

A. Experiment settings

Datasets. We use the following real-world graphs: (1) DBP[2], a knowledge graph that contains entities such as "scientist" or "organization" and relationships among entities such as "worksAt"; (2) OAG, a fraction of the open academic graph[3], where nodes refer to *e.g.,* papers, authors, organizations, and an edge can be "cite" or "affiliatedTo"; (3) Yelp[4], a graph with nodes as users and local services (*e.g.,* plumbers, restaurants), and edges such as "friendWith" and "reviews".

We also report the sizes of individual scope graphs that are induced by specific types (*e.g.,* 'Data Mining' from OAG denotes a scope graph with nodes having a topic 'Data Mining'). The details of these datasets and scope graphs are summarized in Tables II and III, respectively.

[2]https://wiki.dbpedia.org/develop/datasets

[3]https://www.openacademic.ai/oag/

[4]https://www.kaggle.com/yelp-dataset

Data	Constraint (support/confidence)	Transformation rules
Music	If a music genre v^* has derivative "New_Age_Music", its origin is "Blues". (249/0.99)	$\phi(v^*.origin) \longmapsto a(a \neq \text{"}Blues\text{"})$
Transport	If two transportation tools v_1^* and v_2^* are related, they have the same manufacturer. (705/0.88)	$\phi(v_1^*.manufacturer) \longmapsto a(a \neq a_1^*)$
UserGroup1	If users v_1^* and v_2^* friend each other and have the same ratings, and v_1^* has score "5", then v_2^* also has score "5". (157/1.0)	$\phi(v_2^*.\text{score}) \longmapsto a(a \neq \text{"}5\text{"})$

TABLE IV: Examples of Graph Data Constraints

Error Generation. We implemented a configurable error generator that enhances BART [2], a tool to introduce erroneous attribute values to real graphs. Following [2], it introduces two classes of errors to node attribute values:

(1) *Detectable errors* refer to violations of a set of data constraints Σ [10]. We invoke the algorithm in [9] to discover Σ. The generator then injects errors to the nodes that match their "If" condition ("node matches") to violate the constraints. For DBP, OAG, and Yelp and their scope graphs, we have discovered 89, 21, and 112 constraints with 1000, 10, and 20 as minimum support (number of node matches) and 0.9, 0.8, and 0.85 as minimum confidence (the node matches that also satisfy the consequent), respectively. Three constraints (with general transformation rules) are illustrated in Table IV.

(2) *Random errors* are generated from multiple scenarios including misspelling, outlier values (both string and numerical values), missing values ('null'), and random string disturbance. We ensured that injecting these erroneous values alone are not leading to violations of the data constraints Σ.

The generator is controlled by node error rate (the probability that a node is erroneous, set as 0.01 by default), attribute error rate (the probability that an attribute is selected to pollute; set as 0.33 by default), and detectable rate (the probability that an error is a detectable; set as 0.5 by default). We also retain the transformations Ψ that are used to inject the errors for graph augmentation. For each graph, we randomly partition the nodes to obtain 6 folds for training examples, 1 fold for the validation set, and 3 folds for testing nodes (see Table III).

Algorithms. We implemented GEDet (including graph augmentation and adversarial error detection), and compare its performance with 8 baselines categorized as follows.

(1) *Variants of* GEDet, including (a) GEDet-C, a simpler GAN-based alternative that directly differentiates the nodes in G_H and G_L by learning a discriminator over synthetic encodings X_S (from G_H) and original counterpart X_L (from G_L) without graph autoencoders; (b) GEDet-B, an alternative based on few-shot learning, which replaces the adversarial detection with a GCN-based classifier; and (c) GCN, a variant of GEDet-B that learns a semi-supervised node classifier with GCN, without graph augmentation and adversarial training.

Dataset	Metrics	VioDet	OutlierDet	Hybrid	Alad	Raha	GCN	GEDet-B	GEDet-C	GEDet
Agent	P	**0.99**	0.10	0.17	0.39	0.38	0.40	0.50	0.43	0.44
	R	0.20	0.17	0.33	0.37	0.65	0.33	0.33	0.50	**0.67**
	F	0.33	0.13	0.22	0.38	0.47	0.36	0.40	0.46	**0.53**
Music	P	**0.50**	0.17	0.27	0.11	0.21	**0.50**	0.23	0.40	0.24
	R	0.27	0.13	0.50	0.25	0.67	0.17	0.67	0.33	**0.83**
	F	0.35	0.15	0.35	0.15	0.32	0.25	0.34	0.36	**0.37**
Transport	P	**0.27**	0.17	0.21	0.22	0.24	0.22	0.22	0.22	0.24
	R	0.13	0.12	0.32	0.90	0.12	0.88	0.91	0.97	**0.98**
	F	0.17	0.14	0.26	0.35	0.16	0.35	0.36	0.36	**0.39**
Species	P	0.85	0.35	0.35	0.13	0.40	0.57	0.51	**0.98**	0.91
	R	0.24	0.32	0.45	**0.89**	0.60	0.35	0.40	0.43	0.48
	F	0.38	0.34	0.40	0.23	0.48	0.43	0.45	0.60	**0.63**
Data Mining	P	0.26	0.10	0.24	0.23	0.50	0.35	0.58	**0.98**	**0.98**
	R	0.30	0.24	0.43	**0.77**	0.43	0.74	0.48	0.44	0.45
	F	0.28	0.14	0.31	0.35	0.47	0.47	0.53	0.61	**0.62**
Machine Learning	P	0.24	0.08	0.26	0.23	0.62	0.63	0.57	**0.97**	**0.97**
	R	0.27	0.28	0.50	0.40	0.45	0.43	**0.54**	0.43	0.45
	F	0.25	0.13	0.34	0.30	0.52	0.51	0.55	0.59	**0.62**
UserGroup1	P	0.33	0.56	0.31	0.27	0.63	0.51	0.53	**0.81**	0.78
	R	0.55	0.30	**0.67**	0.55	0.60	0.52	0.58	0.53	0.64
	F	0.41	0.39	0.42	0.36	0.62	0.52	0.55	0.64	**0.70**
UserGroup2	P	0.31	0.51	0.28	0.27	0.59	0.66	0.34	0.92	**0.96**
	R	0.54	0.30	0.65	**0.73**	0.56	0.33	0.70	0.43	0.45
	F	0.39	0.38	0.39	0.39	0.57	0.44	0.46	0.59	**0.61**

TABLE V: Performance of Error Detection. **Bold**: best result; Underlined: second best.

(2) *Anomaly detection*, including (a) OutlierDet, which computes a local outlier factor [4] to node features and quantify the degree of being an outlier in k nearest neighbors; and (b) Alad [17], a framework that measures normality of the nodes by considering both the topological structures of the graph and attribute distribution estimation within local context of nodes.
(3) VioDet, a constraint-based error detection that detects errors as the union of the violations of a set of data constraints Σ mined from the original datasets.
(4) Ensemble approaches that combine multiple detection methods: (a) Raha [18] is a state-of-the-art method to detect errors in relational data. It configures a library of built-in detectors *e.g.,* outlier detection, to generate error detection strategies; (b) Hybrid, a method that adopts the Union-all strategy [1] and takes the union of the detected erroneous nodes from OutlierDet and VioDet to improve the recall.

Configuration. We use consistent settings for fair comparison. (1) For OutlierDet and Alad, we optimized model hyperparameters to achieve their best performance. As Alad ranks nodes and is evaluated by AUC-PR curve [17], we applied the default setting to learn Alad, selected the thresholds that enable its best performance in terms of AUC-PR curve, and derived anomalies as erroneous nodes. (2) We used the same set of data constraints Σ for GEDet and its variants, and VioDet (illustrated in Table IV). We use the same settings for variants of GEDet on each dataset. (3) As Raha is designed for relational data, we created one table per node type (with unified schema) and applied Raha to the created table.

Evaluation metrics. We evaluate the performance in effectiveness and efficiency. For effectiveness, we report precision, recall, and F_1-score. Denote Err_d as the set of erroneous nodes detected from the graph, and Err as the set of nodes that are erroneous in the graph. The precision, recall and F_1-score are defined as $\mathsf{P} = \frac{|\mathsf{Err}_\mathsf{d} \cap \mathsf{Err}|}{|\mathsf{Err}_\mathsf{d}|}$, $\mathsf{R} = \frac{|\mathsf{Err}_\mathsf{d} \cap \mathsf{Err}|}{|\mathsf{Err}|}$, and $\mathsf{F} = \frac{2\mathsf{PR}}{\mathsf{P}+\mathsf{R}}$, respectively. For efficiency, we report the training time of the algorithms GEDet, GEDet-B and GCN.

All Experiments were executed on a Unix environment with Intel 2.33GHz CPUs, and 8GB memory. All the algorithms were implemented in Python on Tensorflow. Each experiment was run 5 times and the median results were reported.

Our code and datasets of GEDet are available[5].

B. Experiment results

We first evaluate the effectiveness of GEDet and the baseline algorithms, and the impact of several factors. We then evaluate the training cost of GEDet. In addition, we conduct case studies to evaluate the impact of the quality of data constraints.

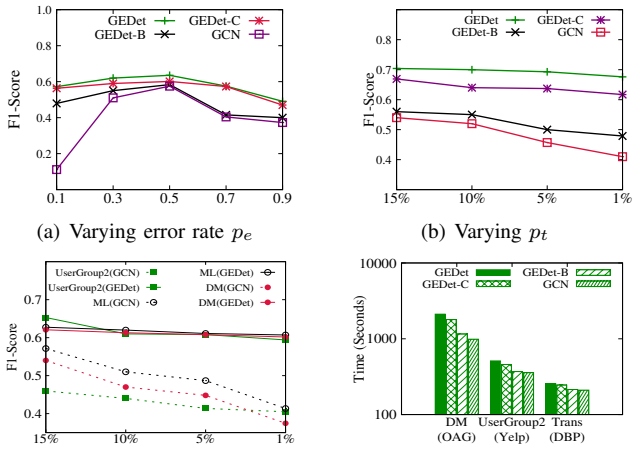
Exp-1: Accuracy of GEDet. We report the accuracy of the methods over all the datasets in Table V. GEDet and its variants (GEDet-C, GEDet-B, and GCN) are learned by using 10% of the training nodes $V_\mathcal{T}$ (summarized in Table III).

(1) There is "no single winner" especially on precision: particular methods may achieve high precision over specific datasets. For example, VioDet is particularly accurate for 'Agent' errors (0.99) due to high quality data constraints.

(2) We take a closer look at the 6 competing methods (all excluding GEDet, GEDet-C and GEDet-B). The low recall of VioDet and OutlierDet suggest the errors are quite diversified. The recall is not improved much by a union-all strategy (Hybrid). Alad is able to improve highest recall over some cases, but at a cost of low precision and F_1-score. Among these, Raha (which assembles multiple methods) and GCN (graph representation learning) achieve best F_1-scores.

(3) Despite the diversified errors, GEDet, GEDet-C and

[5]https://github.com/sxgcase/GEDet



(a) Varying error rate p_e (b) Varying p_t

(c) Varying p_t (d) Learning cost

Fig. 4: Impact of factors to model performance

GEDet-B achieve either the top or the second best results in precision, recall or F_1-score in almost all the datasets. This is not recognized in individual competing methods. On average, GEDet has an additional gain of F_1-score as 23%, 33%, 21%, 24%, 10% and 15% compared with VioDet, OutlierDet, Hybrid, Alad, Raha and GCN, respectively.

The overall performance of GEDet, GEDet-C and GEDet-B are relatively more stable for different node types and errors compared with competing methods. For example, VioDet has a high precision of 0.99 on Agent, and only achieves 0.24 precision on Machine Learning; similarly for OutlierDet.

(4) For all datasets, GEDet, GEDet-C and GEDet-B achieve a comparable performance. GEDet improves the F_1-score of GEDet-B (resp. GEDet-C) with an average gain of 10% (resp. 3%). These verify the effectiveness of the adversarial module and the need for graph embedding learning of GEDet.

Exp-2: Impact of factors. We next investigate the impact of the factors to GEDet. They include (1) the data imbalance $p_e = \frac{|V^e|}{|V_\mathcal{T}|}$, and (2) the training data ratio $p_t = \frac{|V_\mathcal{T}|}{|V|}$.

Impact of Data Imbalance. Fixing other parameters as default, we vary the imbalance p_e from 0.1 to 0.9 over Machine Learning (OAG). Fig. 4(a) tells us that while all methods achieve better performance over more balanced data, GEDet, GEDet-C and GEDet-B are less sensitive compared with GCN, due to the graph augmentation module can counteract imbalanced training examples. Table V consistently justifies this observation as the fraction of $|V^e|$ varies over different datasets (summarized in Table III): compared with GCN, the methods GEDet, GEDet-C, and GEDet-B are less sensitive.

Varying Training data size. Fixing other parameters as default, we vary p_t over dataset UserGroup1(Yelp) from 15% to 1% and report the result in Fig. 4(b) (the result of VioDet and OutlierDet are insensitive and constantly 0.41 and 0.39, respectively; not shown). While the accuracy decreases for all models as less training data is available, GEDet is the least sensitive (remains a recall above 0.6; not shown). Indeed, it effectively counteracts the impact of lack of examples, with the graph augmentation model that improves recall, and the adversarial module that improves the accuracy.

Figure 4(c) verifies the robustness of GEDet over three other datasets. We found that for all the datasets, graph augmentation improves recall better with more available training examples, while the adversarial learning helps retain a stable accuracy.

Exp-3: Learning cost. We evaluate the learning cost of GEDet-based methods. We set the number of epochs as 200 for all the methods and apply an "early-stop" strategy based on validation performance and make GEDet-C and GEDet terminate early if no improvement is observed within consecutive 40 epochs. As shown in Fig. 4(d), (1) it is quite feasible to learn GEDet. For example, it takes 257 seconds to learn GEDet to achieve a recall at 0.98 over Transportation (DBP); (2) GEDet improves the accuracy of error detection at a cost of small overhead. For example, it introduces on average 13%, 57% and 85% additional cost compared with GEDet-B, GEDet-C and GCN, respectively.

Exp-4: Impact of Data Constraints. We further evaluate the impact of the quality and availability of constraints Σ by comparing GEDet with VioDet and GEDet-N, a variant of GEDet that excludes all transformations induced by Σ.

Method	Data Mining	UserGroup1	Machine Learning
GEDet	0.6209	0.7009	0.6217
GEDet-N	0.6030	0.6813	0.6053
VioDet	0.2800	0.4100	0.2500

As shown in the above table, GEDet can exploit high-quality data constraints when available. For Σ with all constraints that have confidence larger than 0.95, GEDet outperforms GEDet-N with a gain of 2% on $F1$-score. Nonetheless, GEDet-N still outperforms VioDet with a gain at least 27% in 'UserGroup1' and up to 35% in 'Machine Learning (OAG)'. This shows that GEDet is quite robust even when no constraints are available.

We also showcase of real-life errors captured by GEDet that are missed by VioDet and OutlierDet. Fig. 5 illustrates a fraction of Transport. An incorrect value "VW Polo MK5" occurs in the subclass of car model v_2, which should be "supermini" [6]; Similarly for v_3. A constraint states that "if a car is a subsequent work of another and they have the same manufacturer, then they belong to the same subclass". This constraint can capture v_3, but fails to capture v_2, as there is no link between v_1 and v_2. On the other hand, OutlierDet fails to capture v_2 or v_3. GEDet successfully identifies the erroneous nodes v_2 and v_3 in the scope induced by the meta paths from v_2 to v_1 via v_4, following which the correct subclass values are exploited to identify the erroneous nodes.

VII. Related Work

Error detection for graphs. Error detection [1], [5] characterize erroneous values as data outliers [19], [12], [21], [17], or violations of data constraints [5]. In particular, graph data dependencies [10], [16] incorporate both topological and value constraints to detect erroneous nodes. It has been observed that individual methods may achieve high precision but often

[6]https://www.wikidata.org/w/index.php?title=Q758735&action=history

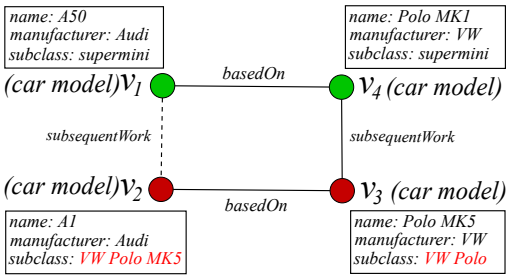


Fig. 5: Case study: Error Detection in DBPedia

low recall even when combined [1], [12]. We integrate transformations to simulate various erroneous scenarios. The graph data augmentation and adversarial learning help improve recall and retain reasonable precision using a few yet diversified erroneous nodes, as verified by our experiments.

Generative Adversarial Networks. GANs have recently been extended to graph learning [29], [8], [7]. LGGAN [8] extends discriminator to generate labeled graphs that preserve important properties from the input graphs. GraphSGAN [7] exploits GANs to improve graph learning, by forcing discriminator to distinguish fake samples injected to node neighbors that are sensitive to skewed label distribution. To the best of our knowledge, GEDet is among the first to incorporate few-shot learning and GANs for graph error detection.

Few-shot Error Detection. For relational data, HoloDetect [12] incorporates few-shot learning to detect data errors. It proposed a policy learning process to generate training data via string transformations. Our work differs from [12] as follows. (1) We enhance few-shot learning to error detection i graphs beyond relational data. (2) Our graph augmentation exploits transformations and neighborhood augmentation beyond string operations. (3) We exploit adversarially learned models to cope with sparse labels. These are not discussed in [12].

VIII. Conclusion

We introduced GEDet, an adversarial and few-shot learning-based error detection framework for graphs. GEDet uses graph augmentation to enhance training examples and neighborhood information, and incorporates graph autoencoders to generative adversarial network to improve the accuracy of error detection. Our experimental study verifies that the graph augmentation and adversarial error detection of GEDet achieve significant gain on accuracy compared with state-of-the-art baselines. A future topic is to enhance GEDet with active learning.

Acknowledgments. This work is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748, DoE under DE-IA0000025, USDA under 2018-67007-28797, and PNNL Data-Model Convergence initiative.

References

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 2016.

[2] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *VLDB*, 2015.

[3] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 2011.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[5] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *SIGMOD*, 2016.

[6] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *NeurIPS*, 2017.

[7] M. Ding, J. Tang, and J. Zhang. Semi-supervised learning on graphs with generative adversarial nets. In *CIKM*, 2018.

[8] S. Fan and B. Huang. Labeled graph generative adversarial networks. *arXiv preprint arXiv:1906.03220*, 2019.

[9] W. Fan, C. Hu, X. Liu, and P. Lu. Discovering graph functional dependencies. In *SIGMOD*, 2018.

[10] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, 2016.

[11] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*, 2018.

[12] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *SIGMOD*, 2019.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[15] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[16] P. Lin, Q. Song, Y. Wu, and J. Pi. Repairing entities using star constraints in multirelational graphs. In *ICDE*, 2020.

[17] N. Liu, X. Huang, and X. Hu. Accelerated local anomaly detection via resolving attributed networks. In *IJCAI*, 2017.

[18] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *SIGMOD*, 2019.

[19] C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.

[20] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 2017.

[21] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng. Anomalous: A joint modeling approach for anomaly detection on attributed networks. In *IJCAI*, 2018.

[22] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv:1712.04621*, 2017.

[23] K. Potdar, T. S. Pardawala, and C. D. Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4), 2017.

[24] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[25] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NeurIPS*, 2016.

[26] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB*, 2011.

[27] C. Sutton, A. McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[28] D. J. Wang, X. Shi, D. A. McFarland, and J. Leskovec. Measurement error in network data: A re-classification. *Social Networks*, 2012.

[29] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. In *AAAI*, 2018.

[30] Y. Wang, Q. Yao, J. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. In *arXiv: 1904.05046*. 2019.

[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[32] R. Zhang, T. Che, Z. Ghahramani, Y. Bengio, and Y. Song. Metagan: An adversarial approach to few-shot learning. In *NeurIPS*, 2018.

[33] Y. Zhang, Y. Xiong, X. Kong, S. Li, J. Mi, and Y. Zhu. Deep collective classification in heterogeneous information networks. In *WWW*, 2018.