GEDet: Detecting Erroneous Nodes with A Few Examples

Sheng Guan
Case Western Reserve University
sxg967@case.edu

Hanchao Ma
Case Western Reserve University
hxm382@case.edu

Sutanay Choudhury
Pacific Northwest National Laboratory
Sutanay.Choudhury@pnnl.gov

Yinghui Wu
Case Western Reserve University
yxw1650@case.edu

ABSTRACT

Detecting nodes with erroneous values in real-world graphs remains challenging due to the lack of examples and various error scenarios. We demonstrate GEDet, an error detection engine that can detect erroneous nodes in graphs with a few examples. The GEDet framework tackles error detection as a few-shot node classification problem. We invite the attendees to experience the following unique features. (1) *Few-shot detection.* Users only need to provide a few examples of erroneous nodes to perform error detection with GEDet. GEDet achieves desirable accuracy with (a) a graph augmentation module, which automatically generates synthetic examples to learn the classifier, and (b) an adversarial detection module, which improves classifiers to better distinguish erroneous nodes from both cleaned nodes and synthetic examples. We show that GEDet significantly improves the state-of-the-art error detection methods. (2) *Diverse error scenarios.* GEDet profiles data errors with a built-in library of transformation functions from correct values to errors. Users can also easily "plug in" new error types or examples. (3) *User-centric detection.* GEDet supports (a) an active learning mode to engage users to verify detected results, and adapts the error detection process accordingly; and (b) visual interfaces to interpret and track detected errors.

PVLDB Reference Format:
Sheng Guan, Hanchao Ma, Sutanay Choudhury, and Yinghui Wu. GEDet: Detecting Erroneous Nodes with A Few Examples. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at http://vldb.org/pvldb/format_vol14.html.

1 INTRODUCTION

Ensuring high-quality graph data is important for applications such as knowledge bases and social networks [6]. The cornerstone task is to detect the nodes with incorrect values ("erroneous nodes"). Various methods have been developed to curate and infer new graph data from correct counterparts [6]. Nevertheless, detecting erroneous nodes in real-world graphs remains challenging.

(1) There are often multiple types of errors. Existing methods [1] are optimized to cope with a single type of error. They may work well for individual scenarios such as violations of data constraints [3] or anomalies [5], yet may not capture multiple types of errors.

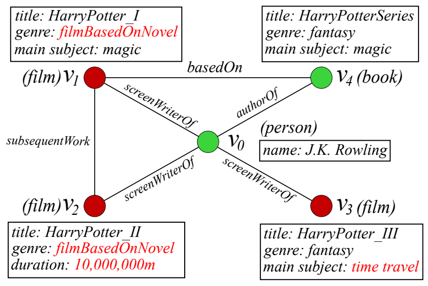


Figure 1: Erroneous Nodes in Knowledge Graphs

(2) One may also use supervised learning to generate a node classifier from labeled nodes ("correct" or "erroneous"). However, it is often hard to obtain a large amount of (manually) labeled examples.

Example 1: Fig. 1 illustrates a fraction of a real-world 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] with different types of errors:

- The genre of film v_1 should be "fantasy";
- 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".

The erroneous nodes v_1 and v_2 can be captured as (1) violation of a data constraint φ [3] that states "if a movie is based on a book (*e.g.,* v_4), then they should have the same genre and main subject", and (2) an outlier with large *duration* [5]. Nevertheless, the node v_3 cannot be captured by either outlier detection (as "time travel" is a common subject value), or the above data constraint (as there is no link between v_3 and v_4). Sequential application of the two detection process overlapped at v_2, yet v_3 remains undetected. □

GEDet. We demonstrate GEDet, a first few-shot learning based Graph Erroneous node DETection system [4]. GEDet only requires a few examples and automatically derives a node classifier to distinguish erroneous nodes from correct ones, and can simultaneously detect multiple types of errors with a desired accuracy.

"Few-shot" detection. To generalize error detection from a few examples, GEDet enables few-shot learning [8] to enrich the examples with similar yet synthetic examples for node classification. It encodes error generation as *transformations*, which are functions that (conditionally) convert correct attribute values to data errors (*e.g.,* anomalies, constraint violations). GEDet derives and maintains transformations from the examples in a built-in library, and performs a graph augmentation process that best approximates

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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

the observed error distribution. Accordingly, it generates synthetic example nodes as well as useful links as enriched examples.

To detect erroneous nodes from correct ones in graphs, GEDet follows representation learning [9] to embed nodes to their low-dimensional vectors (node embeddings), such that the derived labels ("error" or "correct") from the embeddings minimize the classification error given the examples (including synthetic ones). GEDet jointly incorporates attribute-, node-, and topology-level features to derive the classifier. To further reduce the impact of low-quality synthetic examples to the accuracy, GEDet advocates adversarial learning that enforces the classifier to also distinguish synthetic and real examples ("synthetic" or "real") produced by a generator in a "two-player game". These allow GEDet to achieve desirable accuracy (both precision and recall) with little manual effort.

Multi-type error detection. GEDet supports multiple types of errors. By default, it cold-starts with few-shot learning and error detection with a built-in library of transformations. Users can easily declare new error types by "plugging in" examples or transformations. GEDet bookkeeps the transformations and adapts the classifiers to detect errors upon new transformations.

"Human-in-the-loop" error detection. GEDet supports both (1) automated detection that requires little manual effort for system tuning, and (2) interactive detection, which queries the user to verify detected errors and incrementally updates the node classifier via active learning. Moreover, it provides explanations to users on detected errors with relevant transformations and suggested correct values.

GEDet provides user-friendly interfaces for error detection and interpretation [2] with open-source code[3] and a video walkthrough[4].

2 SYSTEM OVERVIEW

2.1 Graphs and Transformations

GEDet framework uses the following specifications.

Graphs. GEDet processes an attributed graph G (where each node is a tuple) in its feature representation (X, A), where (a) X is a matrix of nodes features, and each row X_v of X is a vector encoding of a node tuple v (obtained by *e.g.,* word embedding or one-hot encoding); and (b) A is the adjacency matrix of G.

Examples. An example is simply a node $v \in V$ labeled as 'correct' or erroneous ('error'). GEDet only requires users to provide a few examples $V_{\mathcal{T}}$ for error detection. Optionally, for an erroneous example $v \in V_{\mathcal{T}}$ and its attribute $v.A$ with a wrong value a', a correct counterpart $a*$ can be specified.

Transformations. GEDet characterizes error generation with *transformations*. A transformation (C, ϕ) specifies a condition C (can be empty) and an editing function ϕ defined on node attributes. The transformation (C, ϕ) selects all the nodes that satisfy the condition C, and for each node v, applies ϕ to replace the value $v.A$ to an incorrect value a'. GEDet maintains a built-in library Ψ of transformations that are labeled with its type. By default, it has three types of predefined built-in transformations: "string noise" (randomly modify $v.A$ as a string), "anomalies", and "constraint violations".

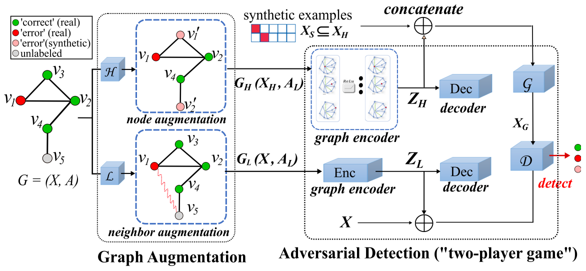


Figure 2: Workflow of GEDet Few-shot Detection

Example 2: GEDet initializes a built-in library Ψ of transformations from *e.g.,* data constraints, domain information or quality rules. For example, given the data constraint φ (Example 1), it registers a transformation T_1 with a condition C that selects nodes with an edge "basedOn" to any "book" node, and $\phi_1(v.genre, \varphi) \longmapsto$ 'filmBasedOnNovel'. Another transformation T_2 can be specified with empty $\phi_2(v.duration) \longmapsto$ a (a>60, 000m) with type "anomalies".

A user may only need to provide two examples $V_{\mathcal{T}} = \{v_1$ ('error'), v_4 ('correct') $\}$ to GEDet (Example 1), The transformations in Ψ (*e.g.,* T_1) will be used to produce synthetic examples. □

2.2 Workflow of GEDet

We start with the major modules and enabling models of GEDet.

Graph augmentation module. This module learns how to generate errors from the examples $V_{\mathcal{T}}$, and in turn enriches examples with synthetic errors and neighbor information. This benefits the follow-up graph representation learning [9], which iteratively updates node embeddings via label propagation.

(1) To generalize from a few examples, GEDet generates synthetic erroneous examples from $V_{\mathcal{T}}$ with an error generative model $\mathcal{H}$, which simulates the generation of various errors characterized by transformations Ψ. The error generation samples a set of correct nodes, and applies $\mathcal{H}$ for each sample to create a set of synthetic erroneous examples (with a synthetic label "error"). The sampling favors correct nodes with erroneous neighbors, to mitigate skewed label distribution (*e.g.,* neighbors that are all correct) that may lead to biased detection. This augments node features X to X_H.

(2) GEDet adopts a link inference model $\mathcal{L}$ to introduce a set of virtual neighbors for the nodes (including examples). The model $\mathcal{L}$ favors to link a node with its reachable, labeled non-neighbors that have similar features, thus enrich its neighbors and mitigates the impact of sparse labels. This enhances adjacency matrix A to A_L.

Error generator $\mathcal{H}$. Given transformations Ψ and a set of erroneous examples $V^e \subseteq V_{\mathcal{T}}$ (where each node $v \in V^e$ has a correct counterpart v^*), the error generative model $\mathcal{H}$ (with learnable weights W) aims to simulate the real transformations from v^* to their erroneous counterparts via a weighted combination of Ψ. To learn $\mathcal{H}$, GEDet solves the following optimization problem:

$$W^* = \arg\max_{W} \log \frac{1}{Z} exp(\sum_{v \in V^e} F(v, \Psi(v^*)))$$

where Z is a normalizer such that the log-term is in $(0, 1]$. The function $F(v, \Psi(v^*)) = \sum_1^n \sum_{\phi_j \in \Psi} w_j \cdot \mathsf{sim}(v.A_i, \phi_j(v * .A_i))$ quantifies the accumulated similarity between the attribute values of each node (an n-ary tuple) v^* and their transformed counterparts.

[2]https://gdet.hcma.repl.co/
[3]https://github.com/CWRU-DB-Group/GDet
[4]https://youtube.com/playlist?list=PL0YKREc7vLdWfX3lXplYvpiklA8zAFEfS

Link inference model $\mathcal{L}$. The model $\mathcal{L}$ (1) samples a set of nodes and their most similar counterparts (determined by a node similarity function [4]), and learns a transition probability for each edge in G, such that each sampled node is more likely to reach their similar counterparts compared with other nodes via a random walk following the transition probability. $\mathcal{H}$ then enhances the adjacency matrix X with new links that connect a (test) node to its top-k non-neighbors with the highest transition probability.

Graph augmentation (illustrated in Fig. 2) yields the following as input for representation learning: (1) $G_H = (X_H, A_L)$ with features of synthetic examples; and (2) $G_L = (X, A_L)$, with the original features X. Both share the augmented topology A_L.

Example 3: Given a correct node v_4 (Example 2) and a transformation T_3 with $(v.main\ subject) \longmapsto (a=\text{'time travel'})$, the error generator $\mathcal{H}$ replaces v_4 with a synthetic erroneous node v_4', where attribute 'main subject' has a value 'time travel'. The link inference $\mathcal{L}$ further identifies nodes v_1 and v_4' as useful virtual neighbors of v_3 due to their common series "Harry Potter" and similar duration (not shown). The augmented data helps GEDet to produce a classifier that captures the erroneous node v_3. □

Adversarial detection module. This module generates a node classifier to detect erroneous nodes. To reduce the impact of synthetic examples, it exploits the principle of *adversarial learning*. The idea is to enforce the classifier to further differentiate synthetic examples from real ones. Specifically, GEDet jointly trains

- a generator $\mathcal{G}$, to "fool" a discriminator $\mathcal{D}$ by simulating the distribution of real labels over the augmented graph G_H; and
- a discriminator $\mathcal{D}$, which aims to classify nodes from G_L and G_H as real or synthetic.

as in a two-player game. By forcing the discriminator to differentiate not only "error" and "correct" but also "synthetic" and "real" labels, GEDet further improves the accuracy of error detection.

Graph autoencoder $\mathcal{Z}$. GEDet uses a graph autoencoder (GAE) $\mathcal{Z}$, a class of graph neural networks, to learn node representations for downstream error detection. Given a graph $G = (X, A)$, a GAE learns an encoding $Z \in \mathbb{R}^{|V| \times d'}$ ($d' \ll d$) of X (by an encoder Enc), from which reconstructing (X, A) is possible (by a decoder Dec). GEDet learns embeddings Z_H and Z_L for $G_H = (X_H, A_L)$ and $G_L = (X, A_L)$, respectively. For G_H, it aims to minimize a reconstruction loss $\min \text{dist}((X_H, A_L), \text{Dec}(\text{Enc}((X_H, A_L))))$, determined by a distance metric dist. The goal for G_L is similarly defined.

Adversarial models ($\mathcal{G}$ and $\mathcal{D}$). GEDet integrates a generative adversarial network (GAN) that consists of a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$. It jointly learns $\mathcal{G}$ and $\mathcal{D}$ to minimize a bi-criteria loss $L(\mathcal{D}) = L^s + \lambda L^u$, where the supervised loss L^s (defined by the cross-entropy errors) quantifies the loss of accuracy on classifying "error" and "correct" examples in a supervised manner. The unsupervised loss L^u (with a balance factor λ=0.5 by default) quantifies the accuracy loss on classifying the real and synthetic examples. The loss is minimized by learning a node embedding matrix $\mathcal{M}$. The matrix $\mathcal{M}$ is then converted via a softmax function to "error" or "correct" class probabilities (see [4] for a formal analysis).

Building Models. GEDet learns $\mathcal{H}$ from Ψ and $V_{\mathcal{T}}$ with L-BFGS [7], and $\mathcal{L}$ with supervised random-walk [2]. It builds graph autoencoder $\mathcal{Z}$ with the layer-wise label propagation paradigm, where the i-th layer updates the embedding of each node by aggregating the embeddings of its neighbors. GEDet applies a co-training algorithm [4] to jointly learn $\mathcal{G}$ and $\mathcal{D}$.

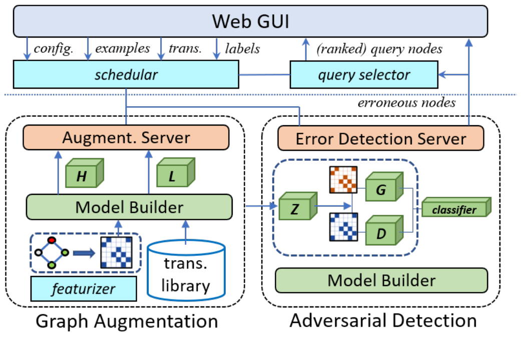


Figure 3: GEDet Architecture (storage layer not shown)

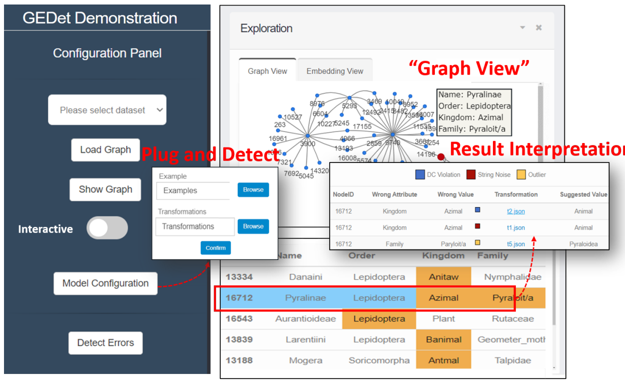
Workflow. GEDet supports automatic and interactive modes. In both modes, it only requires a few examples from users to cold-start.

Automated detection. In this mode, GEDet automatically detects erroneous nodes from scratch without manual tuning effort. (1) In the building phase, GEDet cold-starts to build the models from scratch over given examples $V_{\mathcal{T}}$ via few-shot learning. It initializes transformations including (a) mappings from provided correct values and erroneous ones in $V_{\mathcal{T}}$, (b) available data constraints and quality rules, and (c) random string transformations. (2) The detection phase loads the trained models to assign labels to test nodes of interests (the rest of nodes in G by default). For each test node, GEDet infers its embedding in the classification layer of discriminator $\mathcal{D}$, applies a softmax function to convert the embedding to class probabilities, and chooses the larger one ('error' or 'correct').

Interactive detection. In the interactive mode, GEDet periodically samples detected erroneous nodes and request users for verification. Following active learning, GEDet adopts a query selection policy that favors top-k (k=4 by default) nodes with the least confidence in its most likely label. It queries the user to verify the labels of selected nodes, receives the corrected labels (if any), and incrementally updates the adversarial models $\mathcal{G}$ and $\mathcal{D}$ to improve the accuracy of error detection. Users may also "plug in" new error types as transformations. In this case, GEDet also updates the graph augmentation models $\mathcal{H}$ and $\mathcal{L}$ to incorporate new error types.

2.3 System Architecture

GEDet adopts a three-tier architecture (Fig. 3). (1) The interactive GUI allows users to submit examples and transformation (stored as JSON objects), set detection modes, and inspect errors and interpretations via visual panels (see Section 3). (2) The model tier integrates (a) a *featurizer* that transform input graphs into feature representations, (b) the graph augmentation and adversarial detection modules, (c) a *scheduler* that orchestrates the learning and loading of GEDet models for different detection modes, and (d) a *query selector* that selects detected errors with the selection policy in the interactive mode. (3) The storage tier manages graphs, examples, and built-in transformation libraries.



Name: Pyralinae
Order: Lepidoptera
Kingdom: Azimal
Family: Pyraloit/a

🟦 DC Violation 🟥 String Noise 🟨 Outlier

NodeID	Wrong Attribute	Wrong Value		Transformation	Suggested Value
16712	Kingdom	Azimal	🟦	t2.json	Animal
16712	Kingdom	Azimal	🟥	t1.json	Animal
16712	Family	Paryloit/a	🟨	t5.json	Pyraloidea

	Name	Order	Kingdom	Family
13334	Danaini	Lepidoptera	Anitaw	Nymphalidae
16712	Pyralinae	Lepidoptera	Azimal	Pyraloit/a
16543	Aurantioideae	Lepidoptera	Plant	Rutaceae
13839	Larentiini	Lepidoptera	Banimal	Geometer_moth
13188	Mogera	Soricomorpha	Antmal	Talpidae

Figure 4: User Interface – Automatic Error Detection

3 DEMONSTRATION OVERVIEW

Environment and setup. The GEDet builders and servers are deployed in Google Colaboratory (Colab) environment with Tensorflow libraries and NVIDIA TESLA P100 with 16GB GPU memory. We demonstrate GEDet with the following datasets.

Dataset	$\|V\|$	$\|E\|$	# node types	# edge types	avg. # attr
DBP [5]	2.2M	7.4M	73	584	4
OAG [6]	0.6M	1.7M	5	6	2
Yelp [7]	1.5M	1.6M	42	20	5

These datasets contain both data errors injected by an error generator BART [8], and random errors from multiple scenarios including misspelling, outlier values, and string disturbance.

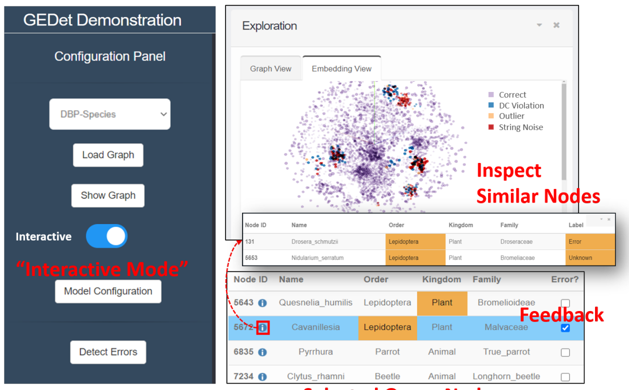
Scenarios. We walk through GEDet with the following scenarios.

Automatic error detection. We invite the users to experience automatic error detection with the user-friendly GUI (Fig. 4). A user can select graph data and submit examples via "Configuration" panel. Using the "Exploration" panel, users will be able to inspect (1) the augmented examples, and enhanced neighbors of specific nodes in the "Graph View" tab, (2) the detected erroneous nodes and their interpretations (*e.g.,* error types, transformations, and suggested correct values), and (3) the clustered visualization of the node embedding $\mathcal{M}$ in terms of error type, in the "Embedding View". The accuracy is reported in the "Performance monitor".

Interactive detection. We also invite users to interact with GEDet to provide guided error detection. Users can switch to "interactive mode" in Configuration panel. GEDet will request users to label a list of detected erroneous nodes. To facilitate the manual labeling of these nodes, GEDet highlights these nodes in the "Graph View", and suggests the top-2 similar nodes with labels are in the "Exploration" panel. The GEDet scheduler resumes learning after user completes verification. Users can also observe the change of decision boundary with the "Embedding View" as more nodes are verified (Fig.5).

Plug-and-detect. A user can also declare and plug in new transformations. Three examples are illustrated below, which are induced from data constraints with validated quality.

[5]https://wiki.dbpedia.org/develop/datasets
[6]https://www.openacademic.ai/oag/
[7]https://www.kaggle.com/yelp-dataset/yelp-dataset
[8]https://github.com/dbunibas/BART



Node ID	Name	Order	Kingdom	Family	Label
131	Drosera_schmutzii	Lepidoptera	Plant	Droseraceae	Error
5653	Nidularium_serratum	Lepidoptera	Plant	Bromeliaceae	Unknown

Node ID	Name	Order	Kingdom	Family	Error?
5643	Quesnelia_humilis	Lepidoptera	Plant	Bromelioideae	☐
5672	Cavanillesia	Lepidoptera	Plant	Malvaceae	☑
6835	Pyrrhura	Parrot	Animal	True_parrot	☐
7234	Clytus_rhamni	Beetle	Animal	Longhorn_beetle	☐

Figure 5: User Interface – Interactive Error Detection

Node type	Constraint (support/confidence)	Transformation (editing function)
Music(DBP)	If a music genre of v^* has derivative "New_Age", its origin is "Blues" (0.99)	$\phi(v^*.origin) \longmapsto a(a \neq \text{"Blues"})$
Transport(DBP)	If two transportation tools v_1^* and v_2^* are related, they have the same manufacturer. (705/0.88)	$\phi(v_1^*.manufacturer) \longmapsto a(a \neq a_1^*)$
UserGroup(Yelp)	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)	$\phi(v_2^*.\text{score}) \longmapsto a(a \neq \text{"5"})$

Performance comparison. We also compare GEDet with 8 state-of-the-art methods (in "Performance" panel) in terms of accuracy, learning cost and detection cost, including: (1) 4 standalone methods covering constraint-based detection, outlier detection, and learning-based classification, (2) 2 state-of-the-art ensemble methods, and (3) 2 variants of GEDet without data augmentation or adversarial detection. while there is no "single winner" from single methods for multi-type errors, GEDet always achieves comparable precision with the best method, and significantly outperforms all the methods in recall. It is also feasible to detect errors in large graphs. For example, it takes on average 350 seconds on model training over OAG, and up to 8 seconds to detect errors, with a gain of precision 30% and recall 35% on average compared with baseline methods.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *VLDB* (2016).
[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*.
[3] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*.
[4] Sheng Guan, Peng Lin, Hanchao Ma, and Yinghui Wu. 2020. GEDet: Adversarially Learned Few-shot Detection of Erroneous Nodes in Graphs. In *IEEE International Conference on Big Data*.
[5] Ninghao Liu, Xiao Huang, and Xia Hu. 2017. Accelerated Local Anomaly Detection via Resolving Attributed Networks.. In *IJCAI*.
[6] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web* (2017).
[7] Charles Sutton, Andrew McCallum, et al. 2012. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning* 4, 4 (2012), 267–373.
[8] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M Ni. 2019. Generalizing from a few examples: A survey on few-shot learning. In *arXiv: 1904.05046*.
[9] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).