An Integrated Graph Neural Network for Supervised Non-obvious Relationship Detection in Knowledge Graphs

Phillipp Müller*, Xiao Qin*, Balaji Ganesan†, Nasrullah Sheikh* and Berthold Reinwald*

IBM Research – Almaden*, IBM Research – India†
{phillipp.mueller, xiao.qin, nasrullah.sheikh}@ibm.com*, bganesa1@in.ibm.com†, reinwald@us.ibm.com*

ABSTRACT

Non-obvious relationship detection (NORD) in a knowledge graph is the problem of finding hidden relationships between the entities by exploiting their attributes and connections to each other. Existing solutions either only focus on entity attributes or on certain aspects of the graph structural information but ultimately do not provide sufficient modeling power for NORD. In this paper, we propose KGMatcher– an integrated graph neural network-based system for NORD. KGMatcher characterizes each entity by extracting features from its *attributes*, *local neighborhood*, and *global position* information essential for NORD. It supports arbitrary attribute types by providing a flexible interface to dedicated attribute embedding layers. The neighborhood features are extracted by adopting aggregation-based graph layers, and the position information is obtained from sampling-based position aware graph layers. KGMatcher is trained end-to-end in the form of a Siamese network for producing a symmetric scoring function with the goal of maximizing the effectiveness of NORD. Our experimental evaluation with a real-world data set demonstrates KGMatcher's 6% to 35% improvement in AUC and 3% to 15% improvement in F_1 over the state-of-the-art.

1 INTRODUCTION

Enterprises are equipped with modern computing power, and excel at storing entities of interest and their relationships generated from daily transactions or operations. Making sense of such linked data has gained increasing importance due to its potential of enabling new services. NORD aims at finding relationships between entities in a knowledge graph where the relationships are not explicitly defined in the data.

One of the first NORD systems [2] was designed to detect credit card fraud and later on gained fame for identifying fake identities in casino businesses. The problem of deciding if two entities share a non-obvious relationship such as "fake identity pair" is challenging. First, the attribute information is an important ingredient to characterize entities. However, the attributes are usually expressed in heterogeneous data structures. Extracting useful features and constructing a unified representation from the attributes through manual feature engineering is tedious and ineffective. Second, two related entities may not share similar attribute properties at all. For example, to trick the registration system, the fake identity is often disguised with totally different demographic and contact information. Instead of predicting solely based on the attribute information, a system should also take the surrounding context into consideration. The "neighbors"

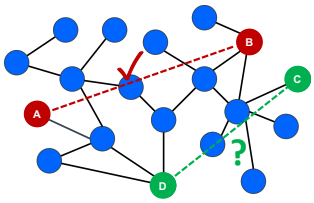

Figure 1: A supervised approach to NORD.

of the entities of interest in the graph may provide useful information for the detector to tap into the truth of a non-obvious relationship. However, modeling such complex context is a non-trivial task. Third, the entities that share non-obvious relationship usually have a high degree of separation in the graph. It is crucial for a system to be capable of capturing the global position information of the entities in the context of the entire graph so that the distanced separation can be identified later on.

Existing solutions rely on handcrafted features to characterize the entities and domain expert defined rules for the detection. With the rapid growth of the graph size and the increasing complexity of the non-obvious relationships, it is tedious and almost impossible to manually maintain such solutions to achieve high effectiveness. Recently, machine learning approaches designed for similar tasks such as entity resolution [5, 9] and graph link prediction [6] are proposed. However, proven by our experimental results, their approaches are only good at tackling certain aspects of the NORD problem and demonstrate limitations in terms of their overall effectiveness. In this work, we propose KGMatcher– an integrated graph neural network-based system for NORD to address the challenges and overcome the limitations of the existing solutions.

KGMatcher Approach. We approach the problem in a supervised machine learning setting as depicted in Figure 1. That is, in addition to the knowledge graph which contains the entities, their attributes and connections, a set of ground-truth labels indicating the existence of the non-obvious relationships between entities is also available. Our goal is to design a machine learning model that can learn from these pairs and discover new pairs.

In its core, KGMatcher is a neural model that automatically extracts important features from the knowledge graph. The features encode the information regarding their entity *attributes*, *neighborhood* and *position* essential for predicting a non-obvious relationship between two entities. KGMatcher consists of three types of neural layers which are connected and can be trained end-to-end. Attribute features are extracted by the *attribute embedding* layers which support heterogeneous attribute types. The dense representations of the entities generated from these layers as well as the edge information are then fed into two stacks of graph layers. The first type of graph layers called *neighborhood* [1] layer focuses on extracting near-by neighborhood information by aggregating their attribute embeddings to the entity of interest. The second type of graph layers named *position* [8] layer focuses on obtaining global position information of the entities

by referring them to a set of sampled anchors. The outputs of the two graph layers are then put together through concatenation to form the final feature vectors of the entities. Finally, KGMatcher's feature extraction network is used in the form of a Siamese network for predicting the existence of a non-obvious relationship between two input entities.

Contributions. It is worthwhile to highlight the following contributions of this work:

(1) We propose a supervised graph neural network-based solution for non-obvious relationship detection called KGMatcher.
(2) We design KGMatcher by adapting and integrating neural architectures for extracting essential NORD features, namely entity *attributes*, *neighborhood* and *position* features.
(3) We demonstrate the effectiveness of KGMatcher using a public dataset. KGMatcher achieves improvement in AUC from 6% to 35% and in F_1 from 3% to 15% over the state-of-the-art.

2 PRELIMINARY

Data Model[1]. Let $G = (V, E)$ denote a knowledge graph where $V = \{v_1, \cdots, v_n\}$ is a set of nodes with each node representing an entity and $E = \{e_1, \cdots, e_m\}$ is a set of edges with each edge $e_k = (v_i, v_j)$ indicating a connection between two entities v_i and v_j. In this work, we assume that the knowledge graph is an undirected graph, i.e., for every $e_k = (v_i, v_j)$, $(v_i, v_j) \equiv (v_j, v_i)$. Let $A = \{a_1, \cdots, a_k\}$ define a set of attributes associated with each entity v_i. An attribute, for example, can be a *date*, an *address*, a *comment*, etc. which means that attributes can be represented in various formats such as numerical, categorical, or text data type. Each entity $v_i = \{x^i_{a_1}, \cdots, x^i_{a_k}\}$ follows the same schema with the attribute types defined by A where $x^i_{a_k}$ denotes the kth attribute value of v_i. In other words, we assume that the entities in the knowledge graph are of the same type, i.e. all the entities share the same attribute types.

Problem Definition. Given two entities v_i and v_j in a knowledge graph G defined above, the goal is to design an algorithm f_G for G that can accurately predict whether or not v_i and v_j share a non-obvious relationship.

In this study, we approach the problem in a supervised learning setting. In addition to the knowledge graph $G = (V, E)$, a ground truth label set $L_G = \{y_1, \cdots, y_r\}$ is also available. A label $y_k = (v_i, v_j)$ where $v_i, v_j \in V$ indicates whether or not there exists a non-obvious relationship between v_i and v_j. Our goal is to design a machine learning model which is able to learn a function f_G from G and L_G that predicts the relationship between two entities presented in G. Following the common practice, the label set is partitioned into train, validation, and test set.

Graph Neural Networks. Graph Neural Networks (GNNs) learn a vector representation of a node from its associated attributes and the graph structure. Modern GNNs [6] adopt a neighborhood aggregation strategy where the representation of a node is learned in an iterative manner by aggregating representations of its neighbors. After k iterations (layers) of aggregation, a node's representation encodes the structural information within its k-hop network neighborhood. Formally, the k-th layer of a GNN is:

$$a^{(k)}_{v_i} = Aggregate^{(k)}\left(\left\{h^{(k-1)}_u | u \in Neighbor(v_i)\right\}\right), \quad (1)$$

$$h^{(k)}_{v_i} = Combine^{(k)}\left(h^{(k-1)}_{v_i}, a^{(k)}_{v_i}\right), \quad (2)$$

[1]The knowledge graph model we adopted in this work can be also seen as a form of the attributed or property graph model referred in the literature.

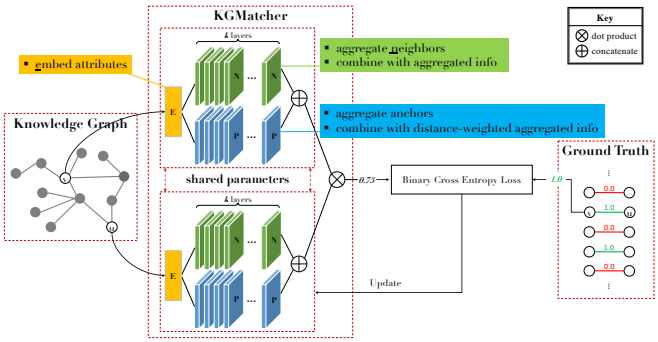

Figure 2: The overview of KGMatcher inference and training.

where $h^{(k)}_{v_i}$ is the vector representation of the node v_i at the k-th iteration. The node's attributes are usually used to initialize $h^{(0)}_{v_i} = Embed(\{x^i_{a_1}, \cdots, x^i_{a_k}\})$ where $Embed(\cdot)$ is an embedding function that obtains a vector representation of the attributes from their raw forms. The exact computation of $Aggregate^{(k)}(\cdot)$ and $Combine^{(k)}(\cdot)$ in GNNs defines their modeling approach. In Section 3.1, we will describe the approaches we introduce to our proposed model and how different models are integrated and trained end-to-end.

3 KGMatcher APPROACH

Overview. We first give an overview of the KGMatcher approach depicted in Figure 2. Each design choice will be discussed in the following sections. The KGMatcher at its core learns a function $f_G(\cdot, \cdot)$ for a knowledge graph G that takes two entities as inputs and produces a score indicating the likelihood of the two entities sharing a non-obvious relationship. KGMatcher characterizes an entity by considering its attributes, its k-hop neighbors' attributes and its position in the knowledge graph. These characterizations are extracted by three types of layers in KGMatcher, namely, **Attribute**, **Neighborhood** and **Position** layer. These layers are connected and trained end-to-end with the goal of producing embeddings of entities which maximize the accuracy of the non-obvious relationship predictions. The function f_G learned by KGMatcher produces a symmetric measure for the input entities which means that $f_G(u, v) \equiv f_G(v, u)$ where $u, v \in V$. The symmetric property is guaranteed by the use of *Siamese network*[4]. The network consists of two identical graph embedding networks which share the same parameters (weight matrices). When measuring two entities, each network takes one of the two inputs and produces the respective embedding. The final score is then computed based on a distance measure between two embeddings.

3.1 Entity Embedding in Knowledge Graph

We introduce the key components in the graph embedding network for entity feature extraction.

Attribute Embedding Layer (E). The attribute information associated with each entity in the knowledge graph provide the central ingredients of the entity. It can be a mixture of structured, semi-structured and/or unstructured data.

By leveraging the existing deep learning based embedding methods, KGMatcher is able to convert arbitrary attributes into a vector representation. First, depending upon the specific type of attribute, one can choose a neural architecture to produce the embedding in an unsupervised or supervised manner. To process "text" type attributes for example, the vector representation can be generated from a pre-trained language model such as XLNet[7]. This unsupervised strategy ensures the generality of the embedding since the pre-trained model is usually obtained from a large general domain corpus. To be able to generate the

embeddings that also maximize the accuracy of the entity matching. KGMatcher allows these neural networks to be easily connected to the rest of the KGMatcher's layers and be adjusted through backpropagation from the feedback on the relationship prediction. The connected embedding networks can be either initialized by the pre-trained parameters (weights) and fine tuned onward or randomly initialized and trained from scratch. Second, KGMatcher finally generates the entity attribute embedding by concatenating each of the embeddings of the whole attribute set and feed it to the next layers.

In our experiment, the attributes are of three types, namely numerical, geo-location (in text string) and categorical type. The numerical types are processed by a normalization layer. The geo-location strings are converted into latitude and longitude numbers. For categorical types, we convert them into one-hot-encodings and embed them using multilayer perceptron neural networks.

Neighbors Aggregation Layer (N). An entity in a knowledge graph is usually not isolated by nature. The connectivity among the entities often indicates additional information which may not be explicitly described by their attributes.

To exploit the natural connections among the entities to capture such "surroundings" signal, we adopt a graph neural architecture as part of the KGMatcher embedding network. The GNNs for this purpose broadly follow a recursive neighborhood aggregation scheme. Each node aggregates embedding vectors of its immediate neighbors to compute its new embedding vector. After k iterations of aggregation, a node is represented by its transformed embedding vector, which captures the surrounding information within the node's k-hop distance. Instead of focusing on embedding nodes from a single fixed graph which is assumed by many prior works, we adopt a spectrum of GNNs – *inductive* GNNs, where only the *aggregate* (Equation 5) and *combine* (Equation 6) for a node are learned. The complexity of such GNNs is usually independent to the size of the graph. They are capable of incorporating unseen nodes and easy to scale.

In particular, we implement *GraphSage*[1] layers as our Neighborhood layers. The *Aggregate* function is formulated as:

$$a_{v_i}^{(k)} = \max\left(\left\{\mathrm{ReLU}\left(W_a^N \cdot h_u^{(k-1)} | u \in Neighbor(v_i)\right)\right\}\right), \quad (3)$$

where W_a^N is a learnable matrix and *max* represents an element-wise max-pooling. The *Combine* function is formulated as a concatenation followed by a linear mapping:

$$h_{v_i}^{(k)} = W_c^N \cdot \left[h_{v_i}^{(k-1)}, a_{v_i}^{(k)}\right], \quad (4)$$

where W_c^N is a learnable matrix. The embedding vector of a node is initialized by its attribute embedding vector obtained from the Attribute layer(s).

Position Measuring Layer (P). The GNNs for neighborhood modeling are good at capturing *local* context. One of the limitations of such modeling approach is their lack of emphasis on the position/location of the embedded node within the broader context of the entire graph. To be able to model the high degree of separation between entities, KGMatcher is equipped with P-GNN layers [8] as Position layers to capture the *global* position information for each entity. The intuition of the P-GNN approach is that the absolute position of a node can be defined by its relative positions to a set of reference nodes in the graph.

Specifically, instead of aggregating information from the immediate neighbors, P-GNN aggregates information for an embedded node from a set of "anchor" nodes. Each P-GNN layer samples a set of anchor nodes as references from the graph and then computes the shortest distances from every embedded node to these sampled anchors to encode a distance metric. Each embedding dimension of a node corresponds to the combined of node embedding and the aggregated information from a specific anchor(s) weighted by the distance metric. The weight is inversely proportional to the distance. KGMatcher stacks multiple Position layers to achieve higher expression power. The aggregation for the lth dimension for v_i at time k is formulated as:

$$a_{v_i^l}^{(k)} = \mathrm{mean}\left(\left\{\mathrm{ReLU}\left(W_a^P \cdot (s(v_i, u) \times h_u^{(k-1)}) | u \in Anchor^l(v_i)\right)\right\}\right), \quad (5)$$

and the combined embedding of v_i at time k is:

$$h_{v_i}^{(k)} = W_t^P \cdot \Big[\cdots, \overbrace{W_c^P \cdot \left[h_{v_i}^{(k-1)}, a_{v_i^l}^{(k)}\right]}^{l\text{th dimension}}, \cdots\Big], \quad (6)$$

where $s(v_i, u)$ computes the weight of an anchor node u to the embedded node v_i, $Anchor^l(v_i)$ returns a set of anchors dedicated for computing the embedding value on the lth dimension of the embedded nodes, *mean* represents an element wise mean-pooling and W_a^P, W_t^P and W_c^P are learnable matrices. The weight computed by $s(v_i, u)$ [8] is inversely proportional to the shortest distance between v_i and u in the graph.

Interaction between N and P Layer. To form a single representation that encodes both neighborhood and position information, a few *merging* design choices are made. One can combine the output embeddings immediately after each N and P layer through concatenation or other element-wise operations. The merged node embedding is then fed separately into the next N and P layer. In our work, to clearly separate the contribution of neighborhood and position signal, we allow the information propagation of the two kinds progress in parallel and only merge the two at the very end through concatenation (depicted in Figure 2).

3.2 KGMatcher Inference and Training

Suppose a pair of entities u and v are labeled with a label $y \in L_G$ using a function $d_y(u, v)$. The goal of KGMatcher is to predict such label y for unseen entity pairs. Specifically, KGMatcher solves the problem via learning an embedding function Φ parameterized by θ, where the objective is to maximize the likelihood of the conditional probability $p(y|\Phi_\theta(u), \Phi_\theta(v))$. Formally, we have the learning objective as:

$$\min_\theta \ \mathbb{E}_{(u,v)\sim L_{G_{train}}} \mathcal{L}(d_z(\Phi_\theta(u), \Phi_\theta(v)) - d_y(u, v)), \quad (7)$$

where $d_z(\cdot, \cdot)$ is a function that predicts the label based on two entity embeddings. Since the relationship we defined is undirected, d_z should then be a symmetric function. A Siamese neural network, depicted in Figure 2 uses the same weights while working in tandem on two different input vectors to compute comparable output vectors which aligns with the required symmetric property. Therefore, we train KGMatcher's entity embedding layers in the form of Siamese network. In our implementation, d_z computes dot product of the two input vectors.

4 EXPERIMENTAL EVALUATION

Setup & Evaluation Method. We evaluate our proposed system with a publicly available dataset by comparing its performance against other baselines. All methods are implemented in PyTorch and trained on a CentOS server with Intel(R) Xeon(R) Gold 6138 @ 2.00GHz CPUs and NVIDIA Tesla P100 GPUs.

Table 1: Test results of different models. ↑ indicates that the higher the score the better the performance. (·) after each score (average±standard deviation) indicates the ranking of the method (vertical comparison) w.r.t the specific evaluation metric. ⋆ indicates that the baseline is implemented by ourselves. The cut-off thresholds of these reported methods are 0.210, 0.445, 0.620, 0.580, 0.440, 0.375 and 0.530 which produce the maximum F_1 score for each respective method.

Method	AUC ↑	F_1 ↑	Precision ↑	Recall ↑
DeepMatcher [5] ⋆	0.6658±0.0351 (6)	0.6755±0.0108 (6)	0.5255±0.0131 (6)	0.9455±0.0000 (2)
GIN [6]	0.7367±0.0329 (3)	0.7400±0.0311 (3)	**0.7036**±0.0415 (1)	0.7818±0.0000 (6)
GCN [3]	0.7080±0.0333 (5)	0.7210±0.0192 (5)	0.6837±0.0346 (2)	0.7636 ±0.0000 (7)
GraphSAGE [1]	0.7633±0.0320 (2)	0.7442±0.0200 (2)	0.6598±0.0311 (4)	0.8545±0.0000 (5)
PGNN [8]	0.7090±0.0337 (4)	0.7255±0.0199 (4)	0.6108±0.0261 (5)	0.8942±0.0195 (4)
PGNN (w/o attributes) [8]	0.5988±0.0424 (7)	0.6668±0.0023 (7)	0.5003±0.0019 (7)	**0.9994**±0.0040 (1)
KGMatcher ⋆	**0.8079**±0.0343 (1)	**0.7660**±0.0239 (1)	0.6676±0.0331 (3)	0.8998±0.0222 (3)

Since our task is binary classification – predict the existence of the relationship between two input entities, we measure the performance of all methods using *receiver operating characteristic* (ROC), *area under the curve* (AUC), *precision*, *recall* and F_1 typical metrics for the evaluation of a binary classification task. We report the average measurements and the standard deviations of all methods on the test set of 100 repetitions.

Dataset. We use the UDBMS *Person*[2] dataset for our evaluation. The original dataset contains 502,529 unique person entities. Each entity has up to 48 attributes. We use 8 attributes – *relation*, *relative*, *spouse*, *child*, *parent*, *partner*, *predecessor*, *successor*, *opponent* and *rival* to build the edges and select another 25 popular attributes in terms of their presenting rate as entity attributes. We use *subject* information to annotate the non-obvious relationships, i.e. two entities sharing the same *subject* have a non-obvious relationship. We further trim down the dataset by only selecting the entities with reasonable amount of attributes and reasonable level of connectivity. Finally, we have 1,294 person entities, 3,480 edges and 316 relationships. The relationship pairs are split into train, validation and test set as the positive samples. The negative samples are uniformly sampled by fixing one of the entities in the positive sample pairs.

Baseline. We compare KGMatcher against 5 other baselines:

- **DeepMatcher** [5] is a supervised deep learning solution which aims to identity pair of data instances that are referring to the same entity based on their attributes.
- **Graph Convolutional Network** (GCN) [3] is a semi-supervised spectral-based graph neural network. It models the graph topology as well as the node attributes.
- **GraphSAGE** [1] is a spatial-based graph neural network which models the graph topology through neighbors aggregation. The aggregated information is based on the node attributes.
- **Graph Isomorphism Network** (GIN) [6] is a graph neural network that generalizes *Weisfeiler-Lehman* test for maximum discriminative power. It also models the node attributes.
- **Position-aware Graph Neural Network** (PGNN) [8] is a generalized spatial-based graph neural network which aims to identify the node position in the context of the the entire graph through sampled anchors.

Effectiveness. Given two entities on the knowledge graph, all methods produce a score [0-1] which can be interpreted as the level of confidence of predicting a non-obvious relationship. As a data analyst, one has to specify a cut-off threshold for the model to give a firm (binary) answer. To simulate this scenario, we measure the effectiveness using *precision*, *recall* and F_1 where the cut-off thresholds must be provided. Since maximizing either *precision* or *recall* can easily done by varying the threshold, we report these measurements by selecting the thresholds that maximized each method's F_1 score. As shown on Table 1, KGMatcher outperformed all baselines in F_1. Although, GIN and PGNN (w/o attributes) perform well either on *precision* or *recall*, their measurements on the other side (*recall* or *precision*) are poor. The unbalanced performance may not be acceptable to many applications. In particular, the model that only considers attributes (DeepMatcher) or global position information (PGNN w/o attributes) performs worse than the ones that model both attributes and some topology of the graph.

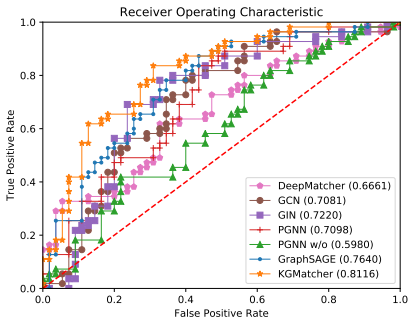

Figure 3: ROC plot of all baselines. (·) after each method in the legend box indicates its corresponding AUC value.

To further evaluate the overall performance of all the methods across different thresholds, we plot the ROC in Figure. 3 and report the AUC value in Table. 1. Our proposed method KGMatcher is significantly better than all the other baselines.

In summary, our experimental evaluation demonstrates KGMatcher's 6% to 35% improvement in AUC and 3% to 15% improvement in F_1 over the other baselines.

[2] http://udbms.cs.helsinki.fi/?datasets/person_dataset

REFERENCES

[1] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the NeurIPS 2017, 4-9 December 2017, Long Beach, CA, USA*. 1024–1034.

[2] Jeff Jonas. 2006. Identity resolution: 23 years of practical experience and observations at scale. In *Proceedings of the SIGMOD 2006, June 26-29, 2006, Chicago, Illinois, USA*. ACM, 718–718.

[3] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the ICLR 2017, April 24-26, 2017, Toulon, France*.

[4] Evgeny Krivosheev, Mattia Atzeni, Katsiaryna Mirylenka, Paolo Scotton, and Fabio Casati. 2020. Siamese Graph Neural Networks for Data Integration. arXiv:cs.DB/2001.06543

[5] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the SIGMOD 2018, June 10-15, 2018, Houston, TX, USA*. 19–34.

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proceedings of the ICLR 2019, May 6-9, 2019, New Orleans, LA, USA*.

[7] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Proceedings of the NeurIPS 2019, December 8-14, 2019, Vancouver, Canada*. 5754–5764.

[8] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware Graph Neural Networks. In *Proceedings of the ICML 2019, June 9-15, 2019, Long Beach, California, USA*. 7134–7143.

[9] Wen Zhang, Kai Shu, Huan Liu, and Yalin Wang. 2019. Graph Neural Networks for User Identity Linkage. *CoRR* abs/1903.02174 (2019).