Related Entity Finding on Highly-heterogeneous Knowledge Graphs

Ridho Reinanda
Bloomberg LP
London, UK
Email: rreinanda@bloomberg.net

Edgar Meij
Bloomberg LP
London, UK
Email: emeij@bloomberg.net

Joshua Pantony
Gradient Boosted Investments
New York City, USA
Email: jrpantony@gmail.com

Jonathan Dorando
Gradient Boosted Investments
New York City, USA
Email: jondorando@gmail.com

***Abstract*—In this paper, we study the problem of domain-specific related entity finding on highly-heterogeneous knowledge graphs where the task is to find related entities with respect to a query entity. As we are operating in the context of knowledge graphs, our solutions will need to be able to deal with heterogeneous data with multiple objects and a high number of relationship types, and be able to leverage direct and indirect connections between entities. We propose two novel graph-based related entity finding methods: one based on learning to rank and the other based on subgraph propagation in a Bayesian framework. We perform contrastive experiments with a publicly available knowledge graph and show that both our proposed models manage to outperform a strong baseline based on supervised random walks. We also investigate the results of our proposed methods and find that they improve different types of query entities.**

I. Introduction

Information about entities and their connections—often encoded in knowledge graphs—is appearing ubiquitously in the context of modern search engines [1, 2]. In a web setting, knowledge graphs are particularly useful for query understanding, presenting entity summaries, and providing explanations for search results [3, 4, 5]. Another popular application of knowledge graphs is to power entity recommendations as well as ranking entities related to a query entity. Existing work in this area mostly focuses on the web search domain, in which the main features of the related entity finding algorithm are typically based on behavioral signals extracted from users' search sessions [6, 7, 8].

Consider a knowledge graph to be a graph $KG = (E, L)$ where E is the set of nodes, i.e., entities, and L the set of directed edges, i.e., relations, where each $l \in L$ is of a certain relationship type. In this paper we consider the task of *domain-specific related entity finding* from the graph KG, i.e., ranking a set of entities E given a query entity e_q within a specific domain. An increasingly prominent requirement for any kind of recommendation is *explainability* of the obtained items [5]. In our setting we realize this by explicitly including the paths between e_q and each ranked entity. The task therefore requires that each entity e in the candidate entity set $E_{e_q} \subseteq KG$ is connected to e_q in KG through at least one path $P_{e_q,e}$ and, furthermore, the entities in E_{e_q} should be ranked based on the relevance of e with respect to the query entity e_q.

Entities can be related in different ways; KGs provide the context on how entities are connected. Examples include public domain KGs such as DBpedia or Freebase in the context of web search [7] or movie and book recommendations [9, 10]. Another example is *governmental impact*. Suppose we have a KG containing companies, people, and other entities as well as several relationship types connecting them. Now consider *Walmart* as the query entity. Assuming that there are paths between *Walmart* and a number of other company and people entities in the knowledge graph, which one does *Walmart* influence the most? Within this context of *governmental impact*, good related entities for *Walmart* would be entities for which their governmental or political power are affected by *Walmart*. Let us consider political power as a latent value z_e attached to an entity e which might fluctuate because of a query entity (e.g., due to campaign donations). Suppose that related entities are connected through simple paths, i.e., paths with no repeated nodes and no loops that connect the entities in less than k hops, then *Walmart* and the entities in this subgraph (e.g., *Politician-A, Politician-B, Lobbyist-C*) can be connected by a multitude of paths comprising different relationship types. Now suppose that *Politician-A* received donations from *Walmart* and *Lobbyist-C* works with *Walmart*, then both *Politician-A* and *Lobbyist-C* are strongly related to *Walmart* and thus suitable related entities for this context. The connections are strong because the political powers z_A and z_C of *Politician-A* and *Lobbyist-C* will fluctuate heavily with respect to *Walmart*. Meanwhile, *Politician-B*—who only has an indirect connection due to a family member who works there—is not a good recommendation of related entity. The political power z_B of *Politician-B* is less likely to be affected by *Walmart*; i.e., z_B depends less on *Walmart*. Although the task may seem simple for small subgraphs, solving this task in larger, more heterogeneous graphs is not trivial.

Working in the setting introduced above, we run into several challenges. For one, knowledge graphs are inherently heterogeneous, i.e., they contain multiple types of entities and multiple types of relationships between entities. Learning which relationship types are important is already challenging. Moreover, when multiple links are combined into a path, the number of possible link combinations will grow exponentially. Finally, we need a way to aggregate the contributions in the case of multiple paths from a query entity to the ranked entities. Most of the work on heterogeneous graphs only considers a limited number of relationship types, where the number of possible paths can be enumerated and their importance can be learned directly from data [11, 12, 9]. In this paper, we are particularly interested in finding related entities for a large number of heterogeneous path combinations, requiring a different strategy.

We therefore propose two novel methods for the domain-specific related entity finding task. Our first approach is based on learning to rank, in which we extract features from the subgraphs connecting the query entity e_q and related entity e. The intuition is that we can leverage signals such as the path length between entities in combination with other features such as the different relationship types in the subgraph for related entity finding. Our second approach is inspired by Bayesian networks, in which we model the related entity finding in heterogeneous graph through propagation in a probabilistic manner and learn to deal with different relationship types accordingly. We make intermediate predictions from the query entity to intermediate entities at every stage, i.e., making predictions locally, and propagate this prediction to the related entity. In the learning phase, we optimize the weights of each relation type globally within this propagation sequence, taking into account all possible paths in a subgraph. Our approach is unique in the sense that it utilizes shared parameters of conditional probability by relationship type across all subgraphs.

Our main contributions can be summarized as follows. First, we propose two approaches for domain-specific related entity findin in the highly-heterogeneous setting. Second, we perform an in-depth analysis and compare our methods against a strong baseline on a novel domain: *governmental impact*.

TABLE I
GLOSSARY OF THE MAIN NOTATION USED IN THIS PAPER.

Symbol	**Gloss**
KG	a knowledge graph
e	an entity, where e_q is the query entity
l_{uv}	a directed edge/link connecting entity u and v
S	a subgraph of KG, where $S_{e_q e}$ is a subgraph containing the set of all simple paths connecting e_q and target e
$\phi_{S_{uv}}$	features extracted from paths S_{uv}
$\phi_{l_{uv}}$	features extracted from directed edge l_{uv}
$B_{S_{uv}}$	a belief graph derived from the subgraph built from S_{uv}
E	a random variable representing node e in B
Q	a random variable representing node e_q in B
I	a random variable representing node i between e_q and e in the belief graph B
$P(E\|Q)$	the entity relatedness probability of node E given Q
ω	a conditional probability function i.e., $P(E\|D)$ for directly connected entity nodes E and D
Ω	aggregated prediction function to estimate $P(E\|Q)$

II. Related Work

Related work comes in different flavors. We first discuss work on related entity finding in a general setting, then delve into graph-based variants, and finish with a discussion on semantic relatedness. Kang et al. [6] propose a machine-learned entity ranking model which leverages knowledge graphs and user data as signals to facilitate semantic search using entities. Continuing this line of work, Blanco et al. [7] extract signals from a variety of data sources: user search sessions, Twitter, and Flickr to train a learning to rank model that identifies entities related to user queries. Bi et al. [8] pursue recommendation methods that are tailored to entities related to individual user interests. Our approaches are different to these as we do not have access to any behavioral data and thus need to fully exploit the signal provided by the semantics of the highly diverse relationship types in our graph.

As to purely graph-based methods that also do not rely on user interaction data, Bordino et al. [13] propose a method for ranking unexpected, interesting entities by extracting entity networks from two sources of user-generated content.Bordino et al. [14] consider the problem of anticipating user search needs based on their browsing activity. Lao and Cohen [11] focus specifically on path ranking algorithms and introduce the *relational retrieval* task. Instead of learning a weight parameter for each edge type as in previous work, they use one weight per edge sequence. A score is then derived based on a linear combination of labeled edge sequences. Backstrom and Leskovec [12] introduce a supervised method for link prediction based on random walks, biasing the random walk procedure to prefer entities based on known preferences in the training data. Since this method also learns a notion of proximity and obtains excellent results, we include it as baseline in our experiments below. Noia et al. [9] propose a learning to rank method for a recommender system that learns the importance of all possible paths in the graph. As our task of related entity finding in a *highly-heterogeneous* setting is different we can not directly compare to this method.

Finally, estimating entity relatedness is an important task which supports many other downstream tasks. One of its primary uses is to support entity linking: Hoffart et al. [15] propose a model for computing semantic relatedness between two entities based on a weighted set of multi-word key phrases. Most studies that investigate closeness in heterogeneous information networks use simple structures such as direct paths [11, 10]. Huang et al. [16] propose using a so-called *meta-structure*, i.e., a directed acyclic graph of object types with edge types connecting them, to measure proximity between entities. Finally, Hulpus et al. [17] propose a general semantic relatedness measure on knowledge bases. We focus on learning the relatedness within a *recommendation domain*, instead of a general notion of relatedness as the methods mentioned above.

Our work is different from the papers discussed above in the following major ways: (1) we focus on related entity finding in a graph setting, (2) we operate in a highly-heterogeneous setting, and (3) instead of learning each path's importance, we base our predictions on subgraph features with a learning to rank and subgraph propagation framework.

III. Problem formulation

Recall that our primary goal in this paper is to develop a method for domain-specific related entity finding on knowledge graphs. We formally define the task as follows. Given a source entity e_q, rank a set of entities $E \in KG$ with respect to

e_q within a specific domain. We formulate a generic approach to solve this task as estimating the domain-specific relevance between pairs of entities, i.e., $rel(e_q, e)$, ordering them, and aggregating the results.

In order to reduce the search space of all possible entities we first obtain a candidate entity subgraph S. We obtain S from KG following a traversal procedure: given e_q, we retrieve all entities directly related to it using incoming and outgoing links and then perform depth-first traversal to retrieve the next set of candidate entities up to k hops. The query entity, related entities, and all the relations between them form our subgraph S_{e_q}. Pairs of entities are obtained from the query entity and all other entities directly and indirectly connected to it.

We employ two different approaches in this paper to estimate relevance, one of which is to predict the relevance within a learning to rank framework as detailed in Section IV-A. Our second approach makes intermediate recommendation predictions from the query entity to intermediate entities at every stage and propagates this prediction to the related entity; it learns to deal with different relationship types accordingly. We estimate $rel(e_q, e)$ as $P(e|e_q)$, i.e., the probability of recommending an entity e given query entity e_q, through a graph-based inference algorithm as detailed in Section IV-B. Finally, as a baseline, we consider a relevance estimation method based on *graph proximity*, which we adapt to support heterogeneous graphs in Section V-C.

Relationships in a knowledge graph can have additional properties. Consider the following relationship: *IsCampaign-Donor* from a company to a person. For this relationship type a property such as the *donation amount* is typically included in a knowledge graph. Furthermore, such numeric properties can be normalized into weights to indicate the possible strength of the connections, which we refer to as *magnitude*. One way to normalize a property value into weights for a particular relationship type is to divide each quantity by the sum of all quantities originating from the same entity for the same relationship type. For the rest of this paper, we focus on computing the magnitude with source-based normalization; however, our approach is generic and can be extended to incorporate source-based, target-based, or both normalizations.

IV. Methods

In this section, we detail our proposed methods for domain-specific related entity finding.

A. Learning to Rank

For our first method we employ learning to rank and rank each candidate entity based on: $rel(e, e_q) = \Psi(\phi_{e,e_q})$, where Ψ is a machine learned ranking model that makes related entity predictions based on $\phi(S_{e_q,e})$, i.e., the feature representation extracted from subgraphs containing all simple paths connecting related entity e to query entity e_q. $\phi(S)$ is further detailed in Table II. We consider length, magnitude, and type features that are meant to capture different intuitions for prediction such as the fact that direct and/or multiple-path connections are important, but also the fact that certain relationship-type connections are important. Furthermore, these simple intuitions can be combined to form a complex set of features, encoding the overall structure of the subgraph whilst keeping the number of features linear with respect to the relationship types.

TABLE II
FEATURES USED FOR RANKING, EXTRACTED FROM THE SUBGRAPH CONTAINING ALL PATHS CONNECTING THE QUERY AND RELATED ENTITY. *N* INDICATES NUMERIC VECTORS AND *V* INDICATES VECTOR FEATURES WITH BOOLEAN VALUES.

Feature	**Description**	**Type**
`NumPaths`	Number of paths in S_{e_qe}	N
`MaxPathLen`	Longest path length	N
`MinPathLen`	Shortest path length	N
`AvgPathLen`	Average path length	N
`MinPathMagnitudeProd`	Minimum $magnitude_{prod}$	N
`MaxPathMagnitudeProd`	Maximum $magnitude_{prod}$	N
`AvgPathMagnitudeProd`	Average $magnitude_{prod}$	N
`MinPathMagnitudeSum`	Minimum $magnitude_{sum}$	N
`MaxPathMagnitudeSum`	Maximum $magnitude_{sum}$	N
`AvgPathMagnitudeSum`	Average $magnitude_{sum}$	N
`BagRelationTypes`	Types of relations in the paths	V
`BagEntityTypes`	Types of entities in the paths	V

Length features. This feature group is designed to capture the general characteristics of all paths connecting the query and target entity. In particular, we focus on the length of the paths and summarize this subgraph by extracting the number of paths connecting the two entities and the shortest/longest/average path length.

Magnitude features. This feature group aims to capture the strength of the relationship that exists between the two entities. We apply the magnitudes with source-based normalization method described in Section III for each relation property. Then, for a path $p_{e_qe} \in S_{e_qe}$ connecting query entity e_q and entity e we compute the path magnitude by aggregating the strength of connections between the two entities as follows:

$$magnitude_{prod}(q, e) = \prod_{l_{uv} \in p_{qe}} weight(l_{uv}), \qquad (1)$$

where $weight(l_{uv})$ indicates the magnitude of the edge connecting two entities; the magnitudes for the paths are computed by multiplying the strength of edges in the paths. In another variant, we also consider using sum instead of product as a way another way of aggregating the link to compute the path magnitude. Finally, we compute the maximum, minimum, and average of the path magnitudes aggregated by sum and products as our subgraph features.

Type features. This feature group is designed to capture the types of entities and entity relationships that exist between all paths connecting the two entities. For each of these, a boolean feature is extracted to indicate whether the particular entity/relation type is found within the paths connecting the two entities, which is then used as a feature.

B. Subgraph Propagation

Our second approach performs related entity finding through a propagation from one entity node to another, starting with the query entity e_q and ending with the candidate entity e. The related entity probability of adjacent entity nodes are estimated given the current entity and its outgoing relations. We do so by creating a belief graph $B_{P_{e_q e}}$ based on the knowledge subgraph $S_{e_q e}$ for each query-entity pair (e_q, e) and then performing a propagation on this belief network. We design a simple but efficient algorithm by extending belief propagation algorithms from related work [18]. First, we represent each entity node e as a random variable E indicating the related entity finding decision on entity e. The links in the graph indicate a causal dependency relationship between entities. However, given the fact (1) that the query and related entities can be connected by multiple paths and (2) how we construct the subgraph, S will have a tree-like structure. Moreover, the knowledge graph links are directed, reflecting a relation triple $\langle d, r, e\rangle$ denoting source entity d, relationship r, and target entity e. Projecting this onto the belief graph B, node D will become the *parent* of node E.

For inference, we simply instantiate the query node Q on the belief graph $B_{P_{e_q e}}$, assigning it as an observed variable. We then propagate this state to the other nodes, obtaining the probabilities of all other entities in the subgraph. Our extension of the Bayesian network utilizes a *parameterized conditional probability* model that estimates the transitive propagation probability after representing the connection between two adjacent entities l_{de} as features $\phi(l_{de})$. In the following subsections, we further detail our approach. We first provide an overview of the inference and prediction procedure on the belief graph and then we detail how we learn the parameters.

Inference. Let $\Omega(.)$ be the forward propagation function, which applies the conditional probability ω sequentially from the source to target node. The probability of $P(E|Q)$ is reduced to the joint probability $P(Q, E, I)$, where I denotes all intermediate nodes between Q and E. Therefore, following $P(E|Q) = \frac{P(Q,E)}{P(Q)}$, we can compute the conditional probability $P(Q|E)$ as the joint probability $P(Q, E, I_1, .., I_n)$. We further assume *local propagation*, i.e., a node must be affected for it to be able to spread the influence to a neighboring (child) node. With this assumption, any parent node D connected to child node E must be affected, allowing us to avoid computing all the combination of values of the intermediate nodes as the joint probability of the subgraph. We can therefore compute the conditional probability efficiently in a top-down manner, propagating the joint probability from query node Q to entity node E. In this forward propagation, the joint probabilities can be computed recursively as detailed in Algorithm 1. Conditional probabilities are computed for each link by applying $\omega(\phi(l_{de}))$, which effectively produces solely local predictions. The probability is computed incrementally, giving us the joint $P(Q, I_1, .., I_n, E)$ once we reach a target entity E.

Learning. One of the key ingredients to perform inference in a Bayesian network are the *conditional probabilities* which serve as the parameters of the model. In a normal Bayesian network, these parameters θ are typically learned from data. One of the important benefits of working with knowledge graphs is that these parameter values can be shared throughout the whole network, i.e., we only need to learn a single *conditional probability model* ω that encodes the different conditional probabilities between entities d and e directly connected by edge l in the KG. In the following section, we discuss how we learn our parameters from our data.

Algorithm 1 computeProb(B, q, e)

```
Input:   Belief graph $B$, query $q$, candidate entity $e$
Output:  Relatedness probability $P$
 1: if IsRoot(E) then
 2:     return 1.0
 3: else
 4:     $L_{in} \leftarrow getIncomingLinks(E)$
 5:     $C \leftarrow \{\}$; $M \leftarrow \{\}$
 6:     for $edge \in L_{in}$ do
 7:         $c \leftarrow \omega(\phi(edge))$
 8:         $C \leftarrow C \cup c$
 9:         $m \leftarrow computeProb(B, q, edge.src)$
10:         $M \leftarrow M \cup m$
11:     end for
12:     $P \leftarrow causalAggregation(C) \times joint(M)$
13:     // compute the conditional and joint probabiities
14:     return $P$
15: end if
```

The related entity probability between two adjacent entities e and d connected by edge l_{de} is indicated as $P(E|D)$.Instead of learning all values of the parameters $P(E|D)$ for every combination of E and D, we learn a conditional probability model ω, parameterized by edge features, through a gradient-descent optimization procedure. The conditional probability model will be shared across different subgraphs generated from our (e_q, e) pair. Recall that, ω local predictions on adjacent nodes are aggregated into Ω in Algorithm 1. The function *computeProb*, denoted as Ω in Algorithm 2, applies the forward inference procedure that we introduced in the previous subsection. Each link between entity e from its parent d is represented as feature vector $\phi(l_{de})$ and we use the one-hot vector of the relationship type of l and the magnitude of the relation w as the feature value in the vector ϕ. The weight will default to 1.0 if the link does not contain any magnitude information. The relatedness probability $P(E|D)$ between two adjacent entities is estimated as follows:

$$P(E|D) = \omega(l_{de}) = \frac{1}{1 + e^{\theta\phi(l_{de})}}, \qquad (2)$$

that is, the probability of entity e given parent entity d is estimated through a sigmoid function using weights θ and the binary feature vector ϕ extracted from the edge that connect the two entities.

Our optimization procedure to learn the function ω is detailed in Algorithm 2. During training, the prediction $\Omega(x_m)$

Algorithm 2 Learning conditional probability model with L-BFGS.

```
Input:  Training data points $M$
Output: Conditional probability model: $\omega$;
 1: $\theta \leftarrow initializeWeights$
 2: while $notConverged(\omega)$ do
 3:     $\mathcal{L} \leftarrow 0.0$
 4:     for each $m \in M$ do
 5:         $f_m \leftarrow \Omega(m)$ // make prediction for $m$
 6:         $\mathcal{L} \leftarrow loss(y_m, f_m)$ // compute logistic loss
 7:     end for
 8:     $\theta \leftarrow updateWeights(\theta, \mathcal{L}))$
 9: end while
10: return $\omega$ // encapsulates the learned $\theta$
```

for each training instance m is made by propagating evidence from the query to the entities connected to it as follows. We first initialize a weight vector θ for the edge features. Next, we make local predictions on direct relations based on the current weights and compute the predictions for every adjacent pair of entities. Then, we propagate the predictions to the child nodes, and so on. In the event of multiple paths connecting the source and the target entity, we introduce an aggregation function that we detail below.

For updating the weights we use a loss function $\mathcal{L}$ and its derivative using the L-BFGS algorithm [19]. This logistic loss is the most suitable in this case as we aim to compute the posterior probabilities for ranking, not simply binary entity relatedness decisions:

$$\mathcal{L} = \sum_{i}^{N} \log(1 + e^{-y_i \Omega(x_i)}), \tag{3}$$

where y_i is the label for a training instance i converted to probabilities (explained below), and $\Omega(x_i)$ the respective probabilistic prediction on training instance x_i. The function $\Omega(.)$ gives the prediction at training time using the current parameter weights θ and features ϕ. By learning the weights of the relationship through forward inferencing, each relationship type is optimized within its occurrence in the context of other relations in the subgraph between the query and candidate entity.

Our subgraph propagation method expects probabilities $P(E|Q)$ as input during training. With relevance labels $g_{e_q e}$ in our training data denoting the relevance between (e_q, e), we convert the labels to probabilities as follows: $P(E|Q) = \frac{g_{e_q e}}{r}$,which divides *label* by the highest possible label r.

Causal aggregation. Since there can be multiple edges linking to node E, E will have multiple parents. This means that for each entity node with multiple parents, multiple paths—one for each parent node—need to be aggregated and taken into account, which is equivalent to modeling *causal aggregation* [18]. There are different ways to address causal aggregation. With our Bayesian network-like approach, it is not feasible to learn the conditional probabilities with joint causes because: (1) we have multiple belief graphs instead of a single Bayesian network, and (2) it will require a very large amount of data to estimate all the conditional probabilities. To address this issue, we employ the *noisy-OR distribution*, which allows us to compress our conditional probability model [18]. Note that our learning framework is generic and can be extended with other methods for causal aggregation. The noisy-OR distribution has the property that each possible cause (i.e., parent in the graph) can exercise its influence independently. This fits our use-case as we want to accumulate effects from multiple paths when estimating the relatedness. We utilize the noisy-OR distribution to combine evidence from multiple parents. This method is computed as $P(E|D_1, D_2, ..., D_n) = 1 - \prod_i \big(1 - P(D|D_i)\big)$,where i iterates over all parents of D in the belief graph.

V. Experimental Setup

In this section we describe the experimental setup including our data, baseline, evaluation metrics, and parameter settings. We first detail the research questions that drive our experiments.

RQ1 How do our proposed methods and the baseline perform on related entity finding?

RQ2 How does the subgraph propagation method compare against the learning to rank method?

RQ3 How do our methods perform across query entities?

RQ4 Can the subgraph propagation method learn the most important relationship types?

A. Data

Our experimental dataset is based on the LittleSis knowledge graph,[1] which focuses on political data around people and organizations and is commonly used to study *governmental impact*. It contains various object types including *people*, *organizations*, and *locations*. The relationships are grouped into ten main categories: *position*, *student*, *member*, *relation*, *donation*, *service*, *lobbying*, *professional* and *ownership*. Currently, this knowledge graph contains facts about 100,000 entities. We preprocess it by leaving out rare relationship types which only appear less than two times, ending up with 168 relationship types comprising 900,000 relationship instances.

B. Relevance assessments

For our relevance assessments we generate the candidate related entities using the following procedure. First, we randomly sample a number of query entities from all the entities in our dataset. Then, for each query entity, we perform candidate generation with the traversal algorithm described in Section III. We extract subgraphs from the knowledge graph by traversing for a maximum of k hops from the query entity. To limit the size of the query subgraph and the number of candidates we constrain the graph traversal based on the degree of each node. If a node has an in-degree above a threshold ($n = 30$) we will not continue traversing the incoming links, as we assume these to be very generic connections and thus

[1]http://www.littlesis.org

to provide very little signal. Finally, we sample a number of candidate entities from the subgraph to be judged. To make sure we have a representative number of entities in each hop, we sample candidate entities separately for each distance value, i.e., we compute the shortest distance from a candidate entity to the query entity, and sample entities with shortest distance in $k \in 1, 2, 3$. This ensures that we have a number of direct and indirect candidates in our dataset.

We then present each sampled query and candidate pair to assessors to judge the related entity finding quality based on the notion of *governmental impact*. We utilize crowdsourcing to collect our relevance judgments and use CrowdFlower as our annotation platform.[2] More specifically, we design a task in which the assessors have to decide the query entity's impact on the candidate entity. We ask the assessors to judge the impact using a 4-grade relevance level, and instruct them to annotate as follows:

- **Not relevant**: the query entity e_q will have no impact on the target entity e.
- **Somewhat relevant**: the query entity e_q might have an impact on the target entity e, although it might be limited.
- **Relevant**: the query entity e_q will have an impact to the target entity e.
- **Highly relevant**: the query entity e_q will have an obvious and strong impact on target entity e.

We also judge 70 query-candidate entity pairs ourselves and use those as test questions to control the quality of the crowd annotations. CrowdFlower will automatically exclude annotators whose agreements fall below a threshold (set to 0.7 following common CrowdFlower guidelines). In the end, we obtain 1600 judgments of query-candidate entity pairs.

From the crowdsourced assessments of the LittleSis dataset we collect evaluation data for 54 query entities. We perform 5-fold cross-validation experiments in which the models are trained on 4 folds of data and tested on the remaining fold. We average the results across folds and report.

C. Baseline

We compare the performance of our proposed methods—Learning to Rank (LTR) and Subgraph Propagation (SP)—against a baseline based on supervised random walks. For LTR we adopt the Random Forest implementation from scikit-learn [20] in our experiments. As we discussed in Section II, we operate in a highly-heterogeneous KG setting and we require domain-specific related entity finding. Thus, approaches based on manually selecting paths, enumerating and learning path weights, general semantic relatedness, and non-graph features such as [11, 9, 7] are ill-suited as baselines.

A baseline method based on random walks is more fitting because the notion of power z_e attached to each entity (as illustrated above) can be considered as a type of proximity or relevance score similar to PageRank. We detail this baseline in the remainder of this section. Although random walk-based methods are typically used to perform unsupervised recommendations on graph data, we adopt a supervised random walk method based on [12] and extend their method to incorporate parameterized edge weights. That is, we learn different transition probabilities for each edge type in the knowledge graph. Intuitively, this allows for a better approximation of relationship type weights.

[2]http://www.crowdflower.com

Our modified version of the supervised random walk method learns from pairwise preferences of entity recommendation. From the original training data, we generate new pairwise training instances (x_i, x_j) for every pair that satisfies $y_i < y_j$, i.e., the relevance label of instance i is lower than that of instance j with respect to the same source entity. In each step of learning the edge weight parameters, we aim to reduce the number of incorrectly ordered PageRank preference pairs. More specifically, we aim to optimize the following loss function after performing a PageRank computation:

$$\min F(\delta) = ||\delta||^2 + \lambda \sum_{(z_l, z_d) \in Z} h(z_l - z_d), \qquad (4)$$

where δ is the parameter for edge weights, λ the regularization parameter, Z is a list of known PageRank ordering such that $z_l < z_d$, and $h(.)$ is the loss function computed from the pairwise PageRank differences. Following [12], we choose the Wilcoxon-Mann-Whitney (WMW) loss with b set to 0.5: $h(x) = \frac{1}{1+\exp(-x/b)}$,which is differentiable and has been proposed to maximize AUC in [21]. We use the learned edge weights to compute PageRank scores in the heterogeneous graph and use the scores to rank the entities for recommendation.

D. Metrics and significance testing

We evaluate the proposed approaches in a rank-based setting. As our problem can be considered as a form of entity ranking, we use metrics commonly used in document retrieval: precision at m and nDCG@m where $m \in \{1, 3, 5, 10\}$. We compute DCG using: $DCG@k = \sum_{i=i}^{k} \frac{2^{rel_i}-1}{\log_2(i+1)}$,and normalize DCG with the ideal DCG to obtain nDCG. To determine whether the difference in performance between methods is statistically significant we apply the student's paired t-test; we use * to denote $\alpha < 0.1$ and ** for $\alpha < 0.05$.

VI. Results and Discussion

In this section we present the results of our experiments and answer the research questions. We first turn to answering **RQ1** and **RQ2**. Table III details the results of our experiments. Overall, the learning to rank method obtains the best performance both in terms of precision and NDCG. We further observe that both our methods improve upon the supervised random walk baseline in all metrics. We then look at the performance of the learning to rank approach. LTR obtains a 10% improvement over the baseline in terms of P@3 and 5.6% in P@10. In terms of NDCG@10, LTR achieves a significant 4.8% improvement. When we turn to the performance of the subgraph propagation (SP) approach we find that it obtains a 7% improvement over the baseline in terms of P@10. In terms of NDCG@10, SP achieves a significant 4% improvement over the baseline.

TABLE III
RESULTS ON THE LITTLESIS KG.

Method	**P@1**	**P@3**	**P@5**	**P@10**	**NDCG@1**	**NDCG@3**	**NDCG@5**	**NDCG@10**
Supervised Random Walks	0.717	0.731	0.724	0.711	0.699	0.787	0.808	0.835
Learning to Rank	**0.836**	**0.816****	**0.790**	**0.759***	**0.798***	**0.841****	**0.858***	**0.874****
Subgraph Propagation	0.750*	0.786	0.752	0.751*	0.736*	0.817	0.825*	0.854**

In summary, both our proposed methods show their potential for domain-specific related entity finding, obtaining improvements on the LittleSis data with *governmental impact* as the domain. Overall, the learning to rank method obtains higher improvements compared to the subgraph propagation method.

A. Performance across query entities

Here we answer **RQ3**, comparing the performance of the different methods across query entities. First, we want to determine whether LTR achieves the overall improvements by consistently outperforming SP. We do so by comparing the NDCG@10 and P@10 of LTR and SP across query entities on the LittleSis dataset. Table IV shows a detailed result of this contrastive analysis. In terms of NDCG, we observe that SP performs better than LTR on 13 query entities, while LTR wins on 28 query entities; on 13 occasions the performance ends up in a tie. When it comes to precision, we observe that SP performs better than LTR on 14 query entities, while LTR wins on 18. There are 18 occasions where the performance ends up in a tie. This result indicates that the two methods perform differently on different sets of queries, as there are cases where SP substantially outperforms LTR, and vice versa. This observation inspires our next experiment and error analysis below.

Our next experiment concerns different characteristics of the query entities subgraph. The relevance assessments on the LittleSis dataset show that each query entity does not always have an equal distribution of relevant judgments, There are some queries for which the subgraph contains substantially more entities that are judged relevant than the ones that are judged non-relevant. For this particular cases, imbalance makes the actual ranking produced by the different methods less important.

Inspired by the imbalance, we focus our attention on a query segment we define as *balanced queries*: queries with at least the same number of more non-relevant candidates than relevant candidates, i.e., these queries' subgraphs are not dominated by relevant entities. The reason for zooming in on these cases is that in a real-world scenario the rankings of candidate entities on such queries tends to matter more than in the case where almost every candidate in the subgraph is relevant. Table V shows the performance of the different methods on this particular segment. Interestingly, we observe that the subgraph propagation method achieves the best performance in this segment. The improvement in terms of P@10 is significant and of a large magnitude (27%). This again confirms the potential of the subgraph propagation method, since it is

TABLE IV
CONTRASTIVE RESULTS OF LTR VS. SP.

Result	**NDCG@10**	**P@10**
LTR wins	28	18
Ties	13	22
SP wins	13	14

TABLE V
RESULTS USING ONLY BALANCED QUERIES.

Method	**NDCG@10**	**P@10**
Supervised Random Walks	0.709	0.552
Learning to Rank	0.766	0.635
Subgraph Propagation	**0.776**	**0.705****

successful in retrieving the relevant entities when the subgraph also contains a considerable number of non-relevant entities.

B. Model Interpretability

In this section, we answer **RQ4**, focusing on the subgraph propagation method. Recall that one main advantage of our subgraph propagation method is that it can learn the importance of each relationship type within the domain, thus providing us with an interpretable model. This is in contrast with the learning to rank model which learns more generic patterns such as *short paths are more important than long paths*. One way to interpret this propagation model is by looking at the learned weights of each relationship type.

Table VI shows the weights of the relations learned by our propagation algorithm. We only show the top-10 relations in the table, although there are up to 168 distinct relationship types. We find that some of these are less important and would not contribute much to related entity finding within the *governmental impact* domain. The model manages to learn that *campaignDonor-campaignRecipient* is important and it also learns that campaign-related and transactional relations such as lobbying, contractor, and investor are very important, and weights these key relationship types consistenyly higher than other, more arbitrary *person-company* or *company-company* relationships. Similarly, key person-organization/company relations such as *foundingPartner* and *insitutionalInvestor* are considered more important than more arbitrary relations such as *isOrganizationMember*, or social relations such as *closeFriends*. In summary, we conclude that the propagation method can learn to distinguish the important relation types and weight them accordingly. This finding is important because *explainability* is an increasingly important requirement in any

TABLE VI
RELATIONSHIP IMPORTANCE WITHIN THE DOMAIN.

Relation	**Weight**
campaignDonor-campaignRecipient	16.658
campaignRecipient-campaignDonor	15.245
lobbyingClient	3.588
foundingPartner	3.241
directLobbying	2.045
donation	1.887
membership	1.765
contractor	1.622
institutionalInvestor	1.267
client	1.25

ranking scenario. Although the learning to rank method obtains better performance compared to subgraph propagation, it can only produce a generic explanation while the subgraph propagation method can estimate the relatedness probability of each intermediate nodes, providing more interpretable explanations.

VII. CONCLUSION

We have studied the task of domain-specific related entity on highly-heterogeneous graphs and we propose two novel graph-based methods: *learning to rank* and *subgraph propagation*. In our learning to rank method, we extract global characteristics of the subgraph connecting query and related entities and learn a model to rank the candidate entities. In the subgraph propagation method, we treat the subgraphs as a Bayesian network, learn shared parameters of conditional probabilities in a supervised fashion, and infer the relatedness probabilities of candidate entities. Focusing on *governmental impact* as our domain, we have experimented with a publicly available knowledge graph. Our experiments show that our proposed methods achieve a good performance, outperforming a supervised baseline method based on graph proximity. In addition, we find that the subgraph propagation method performs better on queries with balanced subgraph and that it is able to provide natural explanations.

For future work, we are interested in exploring several directions. First, we are currently only using edge features in our subgraph propagation model. Our approach is generic and can be extended to include query, source, and target node features. Second, we would like to experiment with training the model on pseudo-labeled data derived from observed entity associations. Finally, we would like to generate explanations that are more accessible for end users, as in [5].

REFERENCES

[1] J. Pound, P. Mika, and H. Zaragoza, "Ad-hoc object retrieval in the web of data," in *WWW '10*, 2010.

[2] T. Lin, P. Pantel, M. Gamon, A. Kannan, and A. Fuxman, "Active objects: Actions for entity-centric search," in *WWW '12*, 2012.

[3] R. Blanco, G. Ottaviano, and E. Meij, "Fast and space-efficient entity linking for queries," in *WSDM '15*, 2015.

[4] F. Hasibi, K. Balog, and S. E. Bratsberg, "Entity linking in queries: Tasks and evaluation," in *ICTIR '15*, 2015.

[5] N. Voskarides, E. Meij, M. Tsagkias, M. de Rijke, and W. Weerkamp, "Learning to explain entity relationships in knowledge graphs," in *ACL '15*, 2015.

[6] C. Kang, S. Vadrevu, R. Zhang, R. v. Zwol, L. G. Pueyo, N. Torzec, J. He, and Y. Chang, "Ranking related entities for web search queries," in *WWW '11*, 2011.

[7] R. Blanco, B. B. Cambazoglu, P. Mika, and N. Torzec, "Entity recommendations in web search," in *ISWC '13*, 2013.

[8] B. Bi, H. Ma, B.-J. P. Hsu, W. Chu, K. Wang, and J. Cho, "Learning to recommend related entities to search users," in *WSDM '15*, 2015.

[9] T. D. Noia, V. C. Ostuni, P. Tomeo, and E. D. Sciascio, "Sprank: Semantic path-based ranking for top-n recommendations using linked open data," *ACM TIST*, 2016.

[10] X. Liu, Y. Yu, C. Guo, and Y. Sun, "Meta-path-based ranking with pseudo relevance feedback on heterogeneous graph for citation recommendation," in *CIKM '14*, 2014.

[11] N. Lao and W. W. Cohen, "Relational retrieval using a combination of path-constrained random walks," *Mach. Learn.*, vol. 81, no. 1, 2010.

[12] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *WSDM '11*, 2011.

[13] I. Bordino, Y. Mejova, and M. Lalmas, "Penguins in sweaters, or serendipitous entity search on user-generated content," in *CIKM '13*, 2013.

[14] I. Bordino, G. De Francisci Morales, I. Weber, and F. Bonchi, "Anticipating information needs via the entity-query graph," in *WSDM '13*, 2013.

[15] J. Hoffart, S. Seufert, D. B. Nguyen, M. Theobald, and G. Weikum, "Kore: Keyphrase overlap relatedness for entity disambiguation," in *CIKM '12*, 2012.

[16] Z. Huang, Y. Zheng, R. Cheng, Y. Sun, N. Mamoulis, and X. Li, "Meta structure: Computing relevance in large heterogeneous information networks," in *KDD '16*, 2016.

[17] I. Hulpus, N. Prangnawarat, and C. Hayes, "Path-based semantic relatedness on linked data and its use to word and entity disambiguation," in *ISWC '15*, 2015.

[18] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988.

[19] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Math. Program.*, vol. 45, no. 3, Dec. 1989.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *JMLR*, vol. 1, no. 1, 2011.

[21] L. Yan, R. H. Dodier, M. Mozer, and R. H. Wolniewicz, "Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic." in *ICML '03*, 2003.