Refining Node Embeddings via Semantic Proximity

Melisachew Wudage Chekol[1] and Giuseppe Pirrò[2(✉)]

[1] Department of Information and Computing Sciences, Utrecht University, Utrecht, Netherlands
m.w.chekol@uu.nl

[2] Department of Computer Science, Sapienza University of Rome, Via Salaria 113, 00198 Rome, Italy
pirro@di.uniroma1.it

Abstract. There is a variety of available approaches to learn graph node embeddings. One of their common underlying task is the generation of (biased) random walks that are then fed into representation learning techniques. Some techniques generate biased random walks by using structural information. Other approaches, also rely on some form of semantic information. While the former are purely structural, thus not fully considering knowledge available in semantically rich networks, the latter require complex inputs (e.g., metapaths) or only leverage node types that may not be available. The goal of this paper is to overcome these limitations by introducing **NESP** (**N**ode **E**mbeddings via **S**emantic **P**roximity), which features two main components. The first provides four different ways of biasing random walks by leveraging semantic relatedness between predicates. The second component focuses on refining (existing) embeddings by leveraging the notion of semantic proximity. This component iteratively refines an initial set of node embeddings imposing the embeddings of semantic neighboring nodes of a node to lie within a sphere of fixed radius. We discuss an extensive experimental evaluation and comparison with related work.

1 Introduction

Nowadays there is abundant graph data on the Web; from friendship graphs in social networks to knowledge graphs [6]. These graphs can be used for several tasks such as link prediction–whether any two nodes are connected by some edge/relation, node classification–identifying the type of a node, node clustering–grouping the same type nodes into the same category, and so on. Recently, graph embeddings have become a popular means to accomplish those tasks.

The idea is to project nodes or triples [7] of a given graph into a low dimensional vector space while maintaining its structure; as an example, nodes sharing an edge will be nearby in the vector space [2]. Research on graph embeddings focuses on two main types of graphs: homogeneous, having a single type of both nodes and edges (e.g., social relationships of bloggers aka. BlogCatalog [27])–and

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12506, pp. 74–91, 2020.
https://doi.org/10.1007/978-3-030-62419-4_5

heterogeneous, having different types of nodes and edges (e.g., DBpedia [20]). Homogeneous graph embeddings such as LINE [24], node2vec [9], and Deepwalk [17] learn node embeddings by using *random walks* to encode node proximity. While random walk based approaches perform well in several embedding tasks, this result does not directly apply to heterogeneous information networks (HINs) aka knowledge graphs (KGs), partly due to the difficulty to precisely take into account the different kinds of node and edge types in the graph.

Some approaches find biased random walks in HIN by leveraging edge weights (e.g., based on TF-IDF in predicates) that are derived from structural information (e.g., Biased-RDF2Vec [16]). Other approaches, besides the structure, also rely on some form of semantic information (e.g., [5,8]) either as an additional input (e.g, metapaths [5], that is, sequences of node types) or implicit (e.g., alternating the node types [12] traversed during the walk generation according to some likelihood probability). A survey of related literature can be found in [2,25].

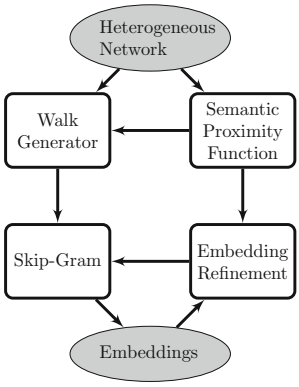

Fig. 1. The **NESP** framework.

While the former are mainly structural-based, thus not fully considering the knowledge available in semantically rich networks (e.g., KGs or HIN), the latter require complex inputs (e.g., it is not clear how to find meaningful metapaths) or only leverage node types where, besides the fact that these may not be available, there is also the need to precisely define the initial stay probability and the number of memorized domains whose optimal values may vary from dataset to dataset. Moreover, none of these approaches can learn (or refine) embeddings where the embeddings of semantic neighbors, even if not directly connected, are constrained to be close together in the vector space.

The goal of this paper is to fill these gaps by introducing a framework called **NESP** (**N**ode **E**mbeddings via **S**emantic **P**roximity), shown in Fig. 1 which *contributes two main components. The first provides four different ways of biasing random walks by leveraging semantic information.* Differently, from approaches using metapaths or similar, where one needs, for instance, to understand the schema-compatibility of the sequence of node types, **NESP** allows to find domain-driven walks when optionally specifying a set of predicates related to a domain of interest. After generating the walks, **NESP** employs the Skip-gram model [15] to learn the node embeddings. *The second component focuses on the notion of semantic proximity*, which identifies the semantic neighborhood of a node that may be different from the structural neighborhood (e.g., considering (directly) connected nodes). For instance, the semantic neighbor of the node (entity) S. Kubrick in DBpedia can be the node M. Forman, which is some hops away from S. Kubrick or another node, which is, for instance of the same type. This second component can be used to refine embeddings found by any existing system. Concretely, it leverages penalty functions [1] to iteratively refine an

initial set of embeddings so that embeddings of the semantic neighbor of a node lie within a sphere of fixed radius centered at the node's embedding. Overall, the contributions of this paper are summarized as follows:

- We introduce a novel approach that leverages edge relatedness to drive the embedding construction. It is much simpler to use edges than nodes (in meta paths) as one needs to guarantee compatibility between node sequences.
- Our approach learns domain-specific embeddings by using some input predicates to specify the domain.
- **NESP** requires simpler input than related approaches (e.g., metapath2vec).
- We introduce an embedding refinement strategy for existing embedding algorithms based on penalty functions and semantic proximity.

The remainder of the paper is organized as follows. We introduce some preliminary definitions in Sect. 2. The **NESP** system is introduced in Sect. 3. Our embedding refinement approach is presented in Sect. 4. We discuss the experimental evaluation in Sect. 5. Related work is treated in Sect. 6. We conclude and sketch future work in Sect. 7.

2 Preliminaries

Heterogeneous Networks. A graph $G = (V, E)$ has a set of nodes V and edges E as well as a function τ that maps nodes and edges into their respective types, formally, $\tau : V \rightarrow T_V$ and $\tau : E \rightarrow T_E$. A Graph G is called *heterogeneous* if nodes (resp. edges) of the graph have different types, i.e., $|T_V| > 1$ (resp. $|T_E| > 1$). When all nodes (resp. edges) of the graph have one type of nodes and one type of edges, i.e., $|T_V| = |T_E| = 1$, then the graph is *homogeneous*. In this work, we are mainly interested in a type of heterogeneous graphs called *knowledge graphs* (KGs) aka heterogeneous information networks (HINs). A knowledge graph is a directed node and edge labeled multi-graph $G = (V, E, U)$ where V is a set of vertices that represent entities, E is a set of predicates and U is a set of triples of the form (s, p, o) representing directed labeled edges where $s, o \in V$ and $p \in E$.

Graph Embedding. A graph embedding model $h : v \rightarrow \mathbb{R}^d$ projects nodes into a low dimensional vector space, where $d \ll |V|$–number of nodes, so that neighboring nodes are nearby in the vector space. At the simplest level, an input to an embedding model consists of an adjacency matrix of a given graph and its output is a vector embedding of each node in the graph so that the distance between two neighboring nodes is much smaller than the ones that are far apart. The dot product between two node embeddings approximates an edge existence between them. After producing an embedding model of a given graph, the model can be used for accomplishing several tasks such as node clustering, node classification, recommendation, and so on. Given their wide array of advantages, graph embeddings have been well researched, we refer the reader to [2] for a survey. A graph embedding model is defined based on a

node similarity and an objective function. Broadly, graph embedding approaches can be divided into two, homogeneous and heterogeneous, based on the kind of graphs they focus on. In this study, we propose an embedding model for heterogeneous networks based on predicate relatedness to find biased random walks and an embedding refinement model based on semantic proximity.

Predicate Relatedness. To find biased random walks **NESP** leverages an existing predicate relatedness measure [18]. Given a knowledge graph $G = (V, E, U)$ and a pair of predicates $(p_i, p_j) \in E$, the relatedness measure is based on the Triple Frequency defined as $TF(p_i, p_j) = \log(1 + C_{i,j})$, where $C_{i,j}$ counts the number of times the predicates p_i and p_j link the same subjects and objects. Moreover, it uses the Inverse Triple Frequency defined as $ITF(p_j, E) = \log \frac{|E|}{|\{p_i : C_{i,j} > 0\}|}$. Based on TF and ITF, for each pair of predicates p_i and p_j we can build a (symmetric) matrix C_M where each element is $C_M(i, j) = TF(p_i, p_j) \times ITF(p_j, E)$. The final predicate relatedness matrix M_R can be constructed such that $Rel(p_i, p_j) = Cosine(W_i, W_j)$, where W_i (resp., W_j) is the row of p_i (resp., p_j) in C_M. We refer to $Rel(p_i, p_j)$ as the relatedness score.

3 NESP: Relatedness-Driven Walk Generation

Unlike previous heterogeneous network embedding approaches that use the types of nodes and metapaths, we use predicates (i.e., edge labels) and their relatedness, to guide walk generation. The input to the walk generator is a graph $G = (V, E, U)$, a predicate relatedness matrix C_M, and a set of input predicates $\mathcal{R}$ that may be used to generate *domain-driven embeddings*, that is, embeddings where some edges are preferred to others during the walk generation. This set if not specified includes all the available predicates. Let $u \in V$ be the current node in a walk, $N(u)$ is the set of its neighbors, $v \in N(u)$ is the *next* node in a walk, $\mathcal{E}(u, v)$ is the set of all predicates linking u and one of its neighbor $v \in N(u)$. Moreover, let $Rel(p_i, p_j)$ be the relatedness score for predicates p_i and p_j. To pick the next node $v \in N(u)$ starting from u we define the following approaches.

Semantic Relatedness Driven Walk. For a given node u and its neighbors $N(u)$, we collect the set of all predicates $\mathcal{E}(u, v)$ between u and its neighbors $\{v \in N(u)\}$. Then, the relatedness between each predicate $p_{uv} \in \mathcal{E}(u, v)$ and all the input predicates $p_j \in \mathcal{R}$ is computed. Once having a relatedness score for each edge (resp. neighbor) this strategy picks as the next node v the one linked to u via the highest relatedness score. When two or more edges have the same relatedness score, one will be selected uniformly at random. The approach is summarized in the following equation:

$$P(v|u,\mathcal{R}) = \begin{cases} 0 \text{ if } |\mathcal{E}(u,v)| = 0 \\ 1 \text{ if } |\mathcal{E}(u,v)| = 1 \\ \varphi(\mathcal{R},\mathcal{E}(u,v)) \text{ otherwise} \end{cases} \qquad (1)$$
$$\varphi(\mathcal{R},\mathcal{E}(u,v)) = \max_{p_j \in \mathcal{R}, p_{uv} \in \mathcal{E}(u,v)} Rel(p_j, p_{uv})$$

Relatedness Driven Jump and Stay Walk. The walks generated by the above approach tend to be biased, indeed, it always picks the node linked via the predicate having the highest relatedness score, thus introducing a form of determinism. To overcome this aspect, we propose relatedness driven jump and stay. For the first M steps of a walk and when the probability of picking the next node is larger than some threshold α, we choose the next node according to Eq. 1 (corresponding to staying); otherwise, we choose a random neighbor (corresponding to jump). We use two parameters: a threshold α for the relatedness score and a step counter M to track the number of nodes visited so far. For each walk, the neighbor v of u is selected according to α and M. For the first M steps (nodes), we choose a node having $\varphi(\mathcal{R},\mathcal{E}(u,v)) > \alpha$. Starting from the $(M+1)$th step, the next node is randomly chosen. By denoting with m the current step, the next node is selected according to the following equation:

$$P(v|u,\mathcal{R},\alpha,m) = \begin{cases} P(v|u,\mathcal{R}) \text{ if } P(v|u,\mathcal{R},\alpha,m) > \alpha \text{ and } m < M \\ x \leftarrow \mathcal{U}(0,1) \text{ otherwise,} \end{cases} \qquad (2)$$

x is a random number chosen from a uniform distribution $\mathcal{U}(0,1)$.

Randomized Relatedness Driven Walk. In this approach, we use a parameter K to specify the percentage of nodes to choose from. As in the previous strategy, we compute the relatedness scores between the input edges and the edges linking the current node u to each of its neighbors v. However, we only consider the top-k highest values that single out a subset of all neighbors (those linked via these top-k predicates). The next node v is selected at random among them. In particular, the higher the relatedness score of an edge, the more likely that it will be selected. Formally v is chosen according to the following distribution:

$$P(v|u,\mathcal{R},K) = \begin{cases} \frac{\varphi(\mathcal{R},p_{uv})}{\sum_{w \in P_r(u,\mathcal{R},K)} w} \text{ if } |\mathcal{E}(u,v)| > 1 \\ P(v|u,\mathcal{R}) \text{ otherwise} \end{cases}$$
$$P_r(u,\mathcal{R},K) = \underset{X' \subseteq \mathcal{X}(u,\mathcal{R}), |X'|=K}{argmax} \sum_{x \in X'} x \qquad (3)$$
$$\mathcal{X}(u,\mathcal{R}) \leftarrow \{\varphi(\mathcal{R},p_j) | \forall p_j \in \mathcal{E}(u,v') \text{ and } \forall v' \in V\}$$

In the denominator of the above equation, we consider only the top-k ranked edges to choose from. To elaborate, $\mathcal{X}()$ contains the set of all relatedness scores between u and its neighbors; and the function $P_r()$ selects the top-k relatedness scores using the *argmax*. Note that if the number of neighbors of u is less than K, then K is changed to the number of its neighbors.

Probabilistic Relatedness Driven Walk. This approach is a variation of the above approach in which instead of considering the top-k neighbors, we choose randomly one of them. In particular, the next node v is sampled from the non-uniform discrete distribution corresponding to the relatedness score according to the following equation:

$$P_t(v|u,\mathcal{R}) = \begin{cases} \frac{\varphi(\mathcal{R},p_{uv})}{\sum_{v' \in V, p_j \in \mathcal{E}(u,v')} \varphi(\mathcal{R},p_j)} \text{ if } |\mathcal{E}(u,v)| > 1 \\ P(v|u,\mathcal{R}) \text{ otherwise} \end{cases} \tag{4}$$

Note that all the above approaches provide much simpler ways of guiding the walks than metapath-based approaches. This is because one has to have precise domain knowledge in picking meaningful metapaths. Not to mention the fact that in schema-rich graphs, it can be the case that no path in the graph satisfies the input metapath. Hence, approaches based on metapaths can only work on graphs that have a simple schema, i.e., a few edge and node types. Moreover, we underline the fact that some of the walk generation strategies are refinements (for instance, probabilistic relatedness driven walk is a variation of randomized relatedness driven walk). The choice of the walk generation strategy depends on many factors including how rich in semantics is the graph, how big is the graph, etc. As an example, if there are node types one can use relatedness driven jump and stay walk. For large graphs, one can pick the semantic relatedness driven walk, which is generally faster as observed in the experiments.

3.1 Learning Node Embeddings

In this section, we describe how nodes of heterogeneous graphs are mapped into vectors in a d-dimensional vector space. For a graph $G = (V, E, U)$ and node $u \in V$, the function h maps nodes into vectors, i.e., $h : u \rightarrow \mathbb{R}^d$ where $d \ll |V|$. Similar to previous approaches, we use the Skip-gram model to learn latent representations of nodes by making use of the walks generated. Skip-gram maximizes the co-occurrence probability among nodes that appear within a given walk of length L. The co-occurrence probability of two nodes u and v in a set of walks W is given by the Softmax function using their vector embeddings e_u and e_v:

$$P((e_u, e_v) \in W) = \sigma(e_u \cdot e_v) \tag{5}$$

In the above equation, σ is the softmax function and $e_u \cdot e_v$ is the dot product of the vectors e_u and e_v. Likewise, the probability that a node u and a randomly chosen node v_j that does not appear in a walk starting from u is given by:

$$P((e_{v_j}, e_u) \notin W) = \sigma(-e_u \cdot e_{v_j}) \tag{6}$$

The negative sampling objective of the Skip-gram model, that needs to be maximized, is given by the following objective function:

$$\mathcal{O}(\theta) = \log \sigma(e_u \cdot e_v) + \sum_{n \in \Gamma} \mathbb{E}_{e_n}[\log \sigma(-e_u \cdot e_n)], \tag{7}$$

where θ denotes the set of all parameters and Γ is the set of negative samples. We use the asynchronous stochastic gradient descent (ASGD) algorithm in parallel to optimize the objective function [12,19].

4 NESP: Embedding Refinement via Semantic Proximity

In this section, we describe the second strategy included in **NESP**. Its goal is to refine node embeddings that have been learned by any existing mechanism both semantic (e.g., JUST [12]) and structural based (e.g., node2vec [10], Deepwalk [17]). The core of the approach is the notion of *semantic proximity*, which for a node identifies a set of semantic neighbors. This definition is very general; the semantic neighbors can be nodes having the same node type (but not necessarily connected), nodes having a similar degree, the edge types, they can be top-k neighbors and so on. The goal of this strategy is to use information about the semantic neighbors (and their initial embeddings) of each node so that *the refined node embeddings are nearby in the vector space*. To elaborate, the refinement strategy allows to modify an initial embedding that may have placed nodes that are semantically neighbors far apart in the vector space.

Semantic Proximity. We mentioned that semantic proximity defines the neighborhood of a node. As mentioned, the notion of neighborhood is general and allows to identify semantically close nodes according to different strategies. We see semantic proximity as a mapping from a node into its semantic neighbors, i.e., $N_K : V \rightarrow 2^{|V|}$. Concretely, an instantiation of N_K can be a function based on semantic relatedness. Given a node u, and a set of input predicates $\mathcal{R}$, describing a domain of interest, N_K can be defined as follows:

$$N_K(u) = P_r(u, \mathcal{R}, K) \tag{8}$$

The definition of $P_r()$ is given in Eq. 3.

Embedding Refinement. Once $N_K(u)$ has been defined, to implement embedding refinement for each node u we consider a sphere with a small radius $r \in \mathbb{R}$ and then constrain the embeddings of the semantic neighbor nodes of u to lie within this sphere. Thus, any two node embeddings within the sphere are at a maximum of $2r$ distance. A small r would place the embeddings of neighboring nodes as close as possible. We mentioned that the initial embeddings to be refined can be computed with any mechanism, which is one of the benefits of our refinement approach. To summarize, under a given semantic proximity, the embedding of u and $v \in N_K(u)$ must be close to each other. Hence, the objective is to minimize the sum of all radii for all v, given the center u, by using Eq. 7 and the cost function given in [1,9].

$$\begin{aligned} \min_{R,C} \quad & \sum_{v \in N_K(u)} \sum_{b \in V} \Big[\log \sigma(e_v \cdot e_b) + \sum_{n \in \Gamma} \mathbb{E}_{e_n} [\log \sigma(-e_v \cdot e_n)] \Big] + \\ & \alpha \sum_{u \in V} r_u^2 + \varphi(u, v, r_u) \\ \text{subject to} \quad & ||e_v - e_u||_2^2 \leq r_u^2,\ r_u \geq 0,\ \forall v \in N_K(u) \text{ and } \forall u \in V \end{aligned} \tag{9}$$

Table 1. Statistics of the datasets used in our experiments.

Graph	$\|V\|$	$\|E\|$	$\|T_V\|$	$\|T_E\|$	#labels
DBLP [11]	∼2M	∼276M	3	4	4
Foursquare [11]	∼30K	∼83K	4	4	10
Yago movies [11]	∼7K	∼89K	4	5	5
PubMed [25]	∼63K	∼245K	10	4	8
AIFB [20]	∼2.4K	∼16K	21	18	4
BGS [20]	∼12K	∼1423K	2	15	3

In the above equation, Γ is the set of negative samples, α denotes a positive weight, the constraint $||e_v - e_u||_2^2 \leq r_u^2$ ensures that (the embeddings of) nodes that have the same semantic proximity as u belong to the sphere of radius r_u and centered at e_u, and R and C denote set of all radii and center embeddings respectively. Besides, $\varphi(u, v, r_u)$ is the penalty function given in Eq. 10. Note that Eq. 9 is a non-convex constrained optimization problem. To convert into an unconstrained optimization problem, we add the following penalty function to the formulation as done in [1].

$$\varphi(u, v, r_u) = \lambda \sum_{u \in V} \sum_{v \in N(u)} f(||e_v - e_u||_2^2 - r_u^2) + \sum_{u \in V} \rho(u) f(-r_u) \qquad (10)$$

In the above formula $\lambda, \rho(u) \in \mathbb{Z}^+$ are parameters which are gradually increased depending on the violation of the constraint. f is a penalty function defined as $f(t) = max(t, 0)$ and controls the violations of the constraints in Eq. 9. We use ASGD to solve the unconstrained optimization problem corresponding to Eq. 9. The optimized solution gives the refined node embedding of v.

5 Experiments

In this section, we report on an empirical evaluation of our approach and comparison with related work. **NESP** has been implemented in Python. Experiments were run on a MacBook P2.7 GHz quad-core processor with 16 GB RAM. The reported results are the average of 5 runs. We used six datasets (see Table 1) from different domains for which the ground truth results were already available.

5.1 Competitors and Parameter Setting

We consider a sample of the most popular and well-performing approaches to compute embeddings in (knowledge) graphs.

– **Deepwalk:** designed for homogeneous graphs, it generates random walks and then feeds them into the Skip-gram model. We set the number of walks per node to $n = 10$, the maximum walk length to $L = 100$ and the window size (necessary for the notion of context in the Skip-gram model) to $w = 10$.

- **node2vec:** this approach improves upon Deepwalk in both the way random walks are generated (by balancing the breadth-first search (BFS) and depth first search (DFS) strategies) and in the objective function optimization (it uses negative sampling). We set the parameter values for n, L, and w to the same values as Deepwalk. Moreover, we also set the parameters p and q, which defines probability of returning to source node and of moving to a node away from the source node, to the best values reported in [9].
- **Metapath2vec:** it takes as input one or more metapaths and generates walks to be fed into the Skip-gram model. We consider the same metapaths used in the evaluation of JUST [12]. For DBLP, Π_1^{DBLP} = A-P-A and Π_2^{DBLP} = A-P-V-P-A linking authors to papers and authors to papers and venues respectively. For Yago movies (mov), Π_1^{mov} = A-M-D-M-A and Π_2^{mov} = A-M-C-M-A representing actors starring in movies with common directors and composers, respectively. For Foursquare (4SQ), Π_1^{4SQ} = U-C-P-C-U representing users checking-in at the same point of interest, and Π_2^{4SQ} = P-C-T-C-P representing points of interest having check-ins at the same timestamp.
- **JUST:** this approach was designed to get rid of metapaths. In doing so, the authors consider probabilities of jump or stay based on the kinds of domains (node types) visited during the generation of a random walk. We set the algorithm parameters to the best values reported in [12].
- **RDF2Vec:** this approach uses graph kernels, extracted using the Weisfeiler-Lehman, to guide walk generation; it assumes equal weights for all edges [20].
- **Biased-RDF2Vec:** this approach focuses on finding domain-specific embeddings by providing a set of input predicates.
- **NESP$_{\mathbf{red}}$:** with this we considered specific subsets of edge labels in the computation of embeddings to evaluate the domain-driven approach. For the DBLP dataset, we considered $\mathcal{R}_1 = \{author\}$ linking a paper to an author, and $\mathcal{R}_2 = \{author, venue\}$ including edges to link a paper to its authors and to the venue it was published. For Foursquare we used $\mathcal{R}_1 = \{perform, locate\}$ and $\mathcal{R}_2 = \{locate, happenedAt\}$ including edges linking users to check-in locations and edges to link the time of checking and the location. For the Yago dataset we used $\mathcal{R}_1 = \{actedIn, directed\}$ and $\mathcal{R}_2 = \{actedIn, coDirector\}$ linking actors to movies and directors and actors to movies co-directed. In these datasets, the set of edges were picked to mimic the link between nodes in the metapaths discussed above. For the datasets AIFB and BGS we use the same predicates used to evaluate *biased-RDF2Vec* [16].

For **NESP**, we considered all the edge types and set the values of n, L, and w as for the other approaches. For each system, we used d=128 as dimension of the node embeddings, and the number of negative samples Γ is set to 10 for all methods in all experiments. For **NESP**, the notion of semantic proximity N_K used in the experiment about embedding refinement is defined as top-k most related nodes via Eq. 8. We only report the best walk strategy, i.e., *probabilistic relatedness driven walk*; the others were slightly worse even though we noticed that *semantic relatedness driven walk* was generally faster. In Table 2, we highlight a summary of the features of the considered systems.

5.2 Node Classification

The goal of this task is to give the label of some test nodes based on the labels of some training nodes. In order to predict the node labels, we use the node embeddings obtained by training **NESP**, node2vec, Deepwalk, JUST, and metapath2vec. Node labels are predicted by using the embeddings as a training input to a one-vs-rest logistic regression classifier. The results of this experiment are given in Fig. 2, where the average (of 5 rounds) Micro-F1 and Macro-F1 scores for classification is shown. For all the datasets and performance metrics (Micro-F1 and Macro-F1), **NESP**'s results are comparable with the baseline, slightly superior in some cases. We observe that as the percentage of training data increases, the performance of all the systems improves, in general. We also observe that when the percentage of training input is above 75%, **NESP** outperforms all the baselines, including $\mathbf{NESP}_{red}$, which makes usage of a subset of predicates. In general, we observe that this variant of **NESP** performs relatively well, especially in the PubMed and AIFB datasets. Unlike all the other systems, JUST's performance slightly decreases as more training input is provided on DBLP. On the other hand, on the datasets PubMed, AIFB and BGS, **NESP** outperforms all the other methods by a higher margin. Note that the results of methpath2vec are not reported for some datasets since metapaths were unavailable and it was difficult to identify meaningful ones. In addition, on the BGS dataset, although not visible from the plot, increasing the size of the samples increases the accuracy. For instance, when we use 10% of the samples, the Micro-F1 is 0.9983 and when we use 20%, it rises to 0.9986, i.e., we obtain a 0.0003 improvement.

Table 2. Walk generation techniques adopted by state of the art systems.

Network type	Approach	Walk guiding		Metapath
		Node type	Edge type	
Homogeneous	Deepwalk	No	No	/
	Node2vec	No	No	/
Heterogeneous	metapath2vec	Yes	No	Yes
	JUST	Yes	No	No
	RDF2Vec	No	No	No
	Biased-RDF2Vec	No	Yes	No
	$\mathbf{NESP_{red}}$	No	Yes	No
	NESP	No	Yes	No

5.3 Node Clustering

The goal of this task is to investigate how similar nodes can be grouped to form communities. We follow the methodology described in Hussein et al. [12]. For each of the graphs, we consider nodes having the same label as being part of the

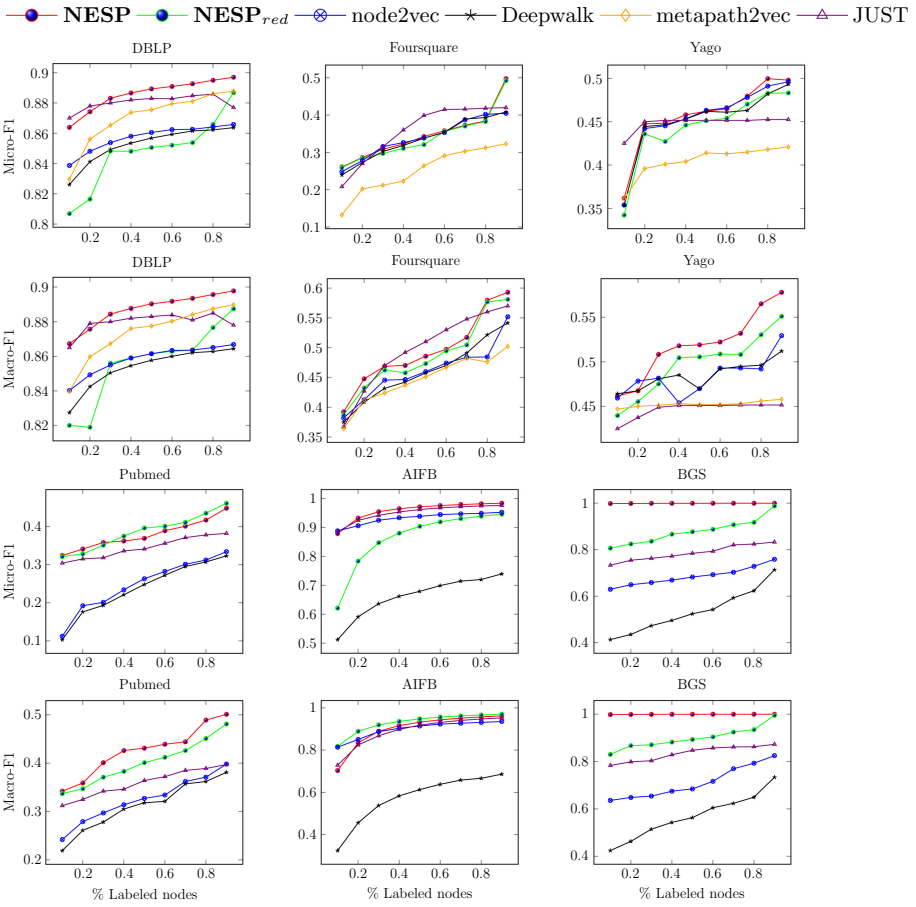

Fig. 2. Node classification results in terms of micro and macro F1.

same community. Then, we fed the node embeddings to the k-means algorithm and evaluated cluster quality by computing the Normalized Mutual Information (NMI). As in the Yago dataset, each movie could be part of more than a class, we provided a classification such that each movie belongs only to one label. This increases the number of possible labels. The results of this experiment are given in Fig. 3. As can be seen, **NESP** outperforms all the baselines on all the datasets. JUST comes second best in terms of clustering performance. Both **NESP** and JUST provide a good indication that one does not need metapaths to outperform state of the art heterogeneous embedding models. In fact, this is clearly visible in the results over the Yago dataset. Interestingly, the performance of homogeneous embedding models, node2vec and Deepwalk is very competitive over the graphs, DBLP and Foursquare. However, on the more schema-rich Yago, their performance degrades showing that they are not well suited for heterogeneous

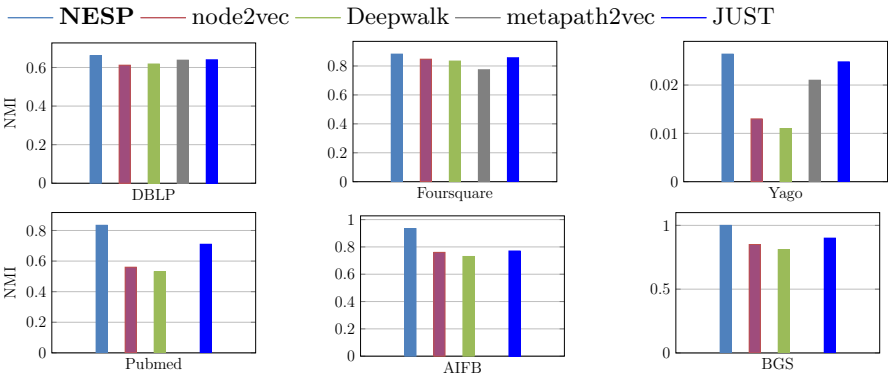

Fig. 3. Clustering results.

graph embedding tasks. In both PubMed and AIFB datasets, which are richer than the other datasets, **NESP** outperforms all the other methods by a very high margin. We also observe that on BGS all approaches performed relatively well with **NESP** reaching the maximum value. For metapath2vec results are not reported for some datasets since metapaths were unavailable and it was difficult to identify meaningful ones.

5.4 Comparison with Domain-Driven Embedding Approaches

In this section we focus on a comparison with *Biased-RDF2Vec* [16], another approach, which focuses on learning domain-driven embeddings. There are crucial differences with our approach. First, *Biased-RDF2Vec* is based on two fixed edge weights; one high, for predicates that *exactly match* the input predicates specifying a domain of interest and one low for the others. On the other hand, **NESP**, relying on the notion of predicate relatedness, covers a more general edge weighting approach. As an example for the input predicate *director*, while *biased-RDF2Vec* matches only this predicate, **NESP** can also assign a high weight to the semantically related predicate like *editor*. In this experiment, we use the same setting (parameters and datasets) used to evaluate *Biased-RDF2Vec* [16]. In particular, we use k-NN (Nearest Neighbor) for node classification with the settings $k = 4$ for AIFB and $k = 10$ for BGS. Below, we also include the results of RDF2Vec [20] that were reported for those datasets. RDF2vec is used as a baseline. Table 3 shows the results of classification accuracy for domain-driven approaches (*biased-RDF2Vec* and **NESP**) as well as structural-driven approach RDF2Vec. The results reported for RDF2Vec are obtained using k-NN. However, higher results were achieved by using SVM (support vector machine), respectively, 93.41% for AIFB and 96.33% for BGS. **NESP** reports a slighter lower value on AIFB and a slightly higher value on BGS with respect to the direct competitor *biased-RDF2Vec*. However, these two datasets were quite specific and limited in the variety of edge types. While Biased-RDF2Vec only assigns

Table 3. Accuracy comparison for classification on AIFB and BGS datasets.

Approach	Dataset	
	AIFB	BGS
Biased-RDF2Vec	99.86%	93.10%
RDF2Vec	88.66%	93.19%
NESP	99.74%	94.04%

Table 4. Evaluation of the embedding refinement strategy.

Approach	Dataset			
	DBLP	AIFB	Foursquare	PubMed
Deepwalk	8%	12%	6%	16%
Node2vec	9%	16%	7%	18%
JUST	4%	8%	4%	6%
NESP	5%	3%	5%	4%

two edge weights (high to the chosen predicate and low to the other) **NESP** offers a broader spectrum of edge weights thus implicitly been able to smooth the difference between predicates of interest for the domain and the others.

5.5 Embedding Refinement

In this set of experiments, the goal was to investigate whether the embedding refinement strategy coupled with some definition of semantic proximity can bring some benefit to embeddings computed by any existing mechanism. We considered embeddings found by: (i) the semantic approaches **NESP** and JUST and (ii) the non-semantic approaches node2vec and Deepwalk. We report experimental results on DBLP, AIFB, Foursquare, and PubMed. The evaluation was carried out as follows. Given a method and a dataset, we computed the initial embeddings, the walks used to learn these embeddings and the set of semantic neighbors $N_K(u)$ for each node u according to Eq. 8. Then, we fed these three inputs and the hyperparameter values to the refinement strategy described in Sect. 4. The output is an updated set of embeddings for each approach and dataset considered. In what follows we focus on the variation of performance in the clustering experiment (see Fig. 3). The results, depicted in Table 4, show the percentage of improvement for each system.

We observe two interesting things. First, the approaches that benefit more from the refinement strategy are Deepwalk and node2vec that completely disregard the semantics of nodes and edges in a graph when computing node embeddings. It shows that refining the embeddings originally found by these approaches by incorporating the notion of semantic proximity is a viable strategy. Second, JUST and **NESP** obtain some improvement, with JUST obtaining a higher margin than **NESP**. This comes as no surprise since the walk generation technique

used by **NESP** (Eq. 4) shares commonalities with the definition of semantic proximity used in the refinement (Eq. 3).

5.6 Parameter Sensitivity

In this section, we investigate the impact of the algorithm parameters related to the Skip-gram model. In particular, we consider the context window size w. We fixed the other parameters to their best values and vary the context window size w from 2 to 12. The results are shown in Fig. 4. We observe that larger values of w lead to better performance. This can be explained by the fact that a larger context window allows to better characterize the notion of neighbors and thus place nodes having a similar neighborhood closer in the embedding space. Nevertheless, when the value of w increases above 10, the benefits are lost, as too long node proximity does not reflect into node similarity. Therefore, we considered $w = 10$ in all previous experiments. We want to stress the fact that **NESP** can capture semantic proximity by favoring the traversal of edge types that are most semantically related to the input edges.

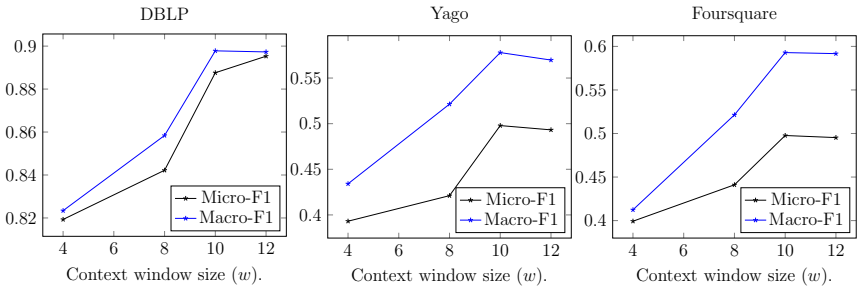

Fig. 4. Impact of the parameter w (context size) for the classification task.

5.7 Discussion

We have observed in the experimental evaluation that **NESP** offers performance comparable or higher than the state of the art in the datasets considered. There are some aspects that make the system more flexible than its competitors. First, **NESP** can be used to learn both domain-specific and general relatedness-driven embeddings. The usage of relatedness allows to implement a sort of semantic proximity during the construction of the walks, which favorably reflects in the fact that nodes with similar neighbors (relatedness-wise) will have closer representations in the embedding space. Second, differently from metapath2vec, **NESP** when used to build domain-specific embeddings, requires a much simpler input, that is, an edge set instead of a complex metapath. We also mention the fact that an input metapath may not even have a counterpart in the underlying data. As compared to domain-driven approaches like biased-RDF2Vec, **NESP**

is more flexible as it considers a broader range of edge weights than the only two weights considered by the former. Another important aspect is that **NESP** (besides those necessary for Word2vec) does not require to set any parameter while biased-RDF2Vec needs to manually assign the two weights and JUST requires to input both the number of domain to memorize (parameter m) and the initial stay probability (parameter α), the choice of which affects the overall quality of the walks generated as well as the embeddings. Finally, the embedding refinement strategy showed to be potentially very useful in combination with simpler approaches like Deepwalk and node2vec besides providing some improvement to the embeddings found by other systems too.

6 Related Work

There are two broad directions in the area of graph representation learning: homogeneous and heterogeneous graph embeddings. Homogeneous graph embedding models include DeepWalk, node2vec, LINE and others. Recently, there is a growing interest in adapting random walk generation techniques from these models to heterogeneous graphs. As a result, IPE [13] (and an earlier version ProxEmbed [14]), JUST, metapath2vec, HIN2Vec [8], PTE [23], ESim [21], HINE [11] and biased-RDF2Vec [16] have been proposed. While IPE uses an interactive path (a directed acyclic graph) for semantic proximity search, SHNE [28], metapath2vec, HIN2Vec, PTE and HINE rely on metapaths. However, generating metapaths is a difficult task and often involves a domain expert [12]. JUST is a heterogeneous graph embedding model which gets ride off metapaths to generate walks. It takes into account the types of nodes in order to compute a jump probability from one node to another when creating random walks. Instead of using node types like JUST, **NESP** uses edge types and their relatedness. Moreover, when giving a specific subset of predicates as input, **NESP** can generate domain-driven embeddings. The approach of [16] also generates domain-specific embeddings. However, it is based on the manual assignment of weights for the predicates given as input while for the others weights are assigned using different strategies (e.g., predicate frequency). **NESP** is more general, domain-independent and does not require an expert for weight assignment as weights are automatically derived from the co-occurrences of predicates in the graph considered.

MetaGraph2Vec [29] uses metagraphs instead of metapaths in order to guide walk generation. However, similar to metapaths, building metagraphs is challenging. Along the same line, RDF2Vec [20] extracts subtree graph kernels from RDF graphs using the Weisfeiler-Lehman method. For each node in a given RDF graph, a subtree up to some depth k is extracted and this subtree is used as a walk to learn node embeddings of RDF graphs using two models: continuous bag of words (CBOW) and Skip-gram. In addition, RDF2Vec provides an alternative way of walk generation using a breadth-first algorithm. Unlike **NESP**, which takes into account the semantics of edges, RDF2Vec picks the next node relying on structural information. Other relevant literature in the area of heterogeneous embeddings [11,22,26]. There exist also semi-supervised approaches that

leverage user feedback to improve the learned embeddings for heterogeneous networks [3]. Away from approaches that are based on local patterns (graph walks, subgraphs and kernels), RDF vector embeddings based on the GloVe (Global Vectors) model have been proposed in [4]. Glove-RDF embedding method creates a global co-occurrence matrix from graphs instead of random walks. For a more detailed discussion on heterogeneous and homogeneous embeddings, we refer the reader to the survey [2,25].

Finally, our notion of semantic proximity, used for embedding refinement, has been inspired by the notion of collective homophily from social networks. In particular [1] used collective homophily to refine embeddings of edges from social networks (including only one edge type). However, our formulation is based on a general notion of semantic neighborhood.

7 Concluding Remarks and Future Work

We described **NESP** a novel approach to heterogeneous graph embeddings including two components. The first features four different walk generation approaches based on predicate relatedness and not only structural information. **NESP** does not require metapaths, node types or other parameters to generate such walks. The second features an embedding refinement strategy that can be applied to embeddings learned by any existing system. It is based on the notion of semantic proximity, which first identifies the semantic neighbors of each node according to any notion of neighborhood (e.g., nodes sharing the same type, nodes linked by specific paths, etc.) and then strive to arrange the positions of a node and those of its neighbor as close as possible in the vector space. The experimental evaluations show that **NESP** is competitive and that the embedding refinement strategy is a viable solution, especially for approaches originally designed for homogeneous networks. Further investigating the refinement strategy and performing experiments on other domains is in our research agenda.

References

1. Bandyopadhyay, S., Biswas, A., Murty, M., Narayanam, R.: Beyond node embedding: a direct unsupervised edge representation framework for homogeneous networks. arXiv preprint arXiv:1912.05140 (2019)
2. Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018)
3. Chen, X., Yu, G., Wang, J., Domeniconi, C., Li, Z., Zhang, X.: ActiveHNE: active heterogeneous network embedding. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI), pp. 2123–2129 (2019)
4. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global RDF vector space embeddings. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 190–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_12
5. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining (KDD), pp. 135–144 (2017)

6. Fionda, V., Pirrò, G.: Querying graphs with preferences. In: Proceedings of the 22nd International Conference on Information and Knowledge Management (CIKM), pp. 929–938 (2013)
7. Fionda, V., Pirrò, G.: Learning triple embeddings from knowledge graphs. In: Proceedings of the 34th Conference on Artificial Intelligence (AAAI), pp. 3874–3881 (2020)
8. Fu, T., Lee, W.C., Lei, Z.: Hin2vec: explore meta-paths in heterogeneous information networks for representation learning. In: Proceedings of the Conference on Information and Knowledge Management (CIKM), pp. 1797–1806 (2017)
9. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD), pp. 855–864 (2016)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of Advances in Neural Information Processing Systems (NIPS), pp. 1024–1034 (2017)
11. Huang, Z., Mamoulis, N.: Heterogeneous information network embedding for meta path based proximity. arXiv preprint arXiv:1701.05291 (2017)
12. Hussein, R., Yang, D., Cudré-Mauroux, P.: Are meta-paths necessary?: revisiting heterogeneous graph embeddings. In: Proceedings of the 27th International Conference on Information and Knowledge Management (CIKM), pp. 437–446 (2018)
13. Liu, Z., et al.: Interactive paths embedding for semantic proximity search on heterogeneous graphs. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1860–1869. ACM (2018)
14. Liu, Z., et al.: Semantic proximity search on heterogeneous graph by proximity embedding. In: Proceedings of 31st Conference on Artificial Intelligence (AAAI) (2017)
15. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Proceedings of the 1st International Conference on Learning Representations (ICLR) (2013)
16. Mukherjee, S., Oates, T., Wright, R.: Graph node embeddings using domain-aware biased random walks. arXiv preprint arXiv:1908.02947 (2019)
17. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: online learning of social representations. In: Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining (KDD), pp. 701–710 (2014)
18. Pirrò, G.: Building relatedness explanations from knowledge graphs. Semant. Web **10**(6), 963–990 (2019)
19. Recht, B., Re, C., Wright, S., Niu, F.: HOGWILD: a lock-free approach to parallelizing stochastic gradient descent. In: Proceedings of Advances in Neural Information Processing Systems (NIPS), pp. 693–701 (2011)
20. Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: RDF2Vec: RDF graph embeddings and their applications. Semant. Web **10**(4), 721–752 (2019)
21. Shang, J., Qu, M., Liu, J., Kaplan, L.M., Han, J., Peng, J.: Meta-path guided embedding for similarity search in large-scale heterogeneous information networks. arXiv preprint arXiv:1610.09769 (2016)
22. Shi, Y., Zhu, Q., Guo, F., Zhang, C., Han, J.: Easing embedding learning by comprehensive transcription of heterogeneous information networks. In: Proceedings of International Conference on Knowledge Discovery & Data Mining (KDD), pp. 2190–2199 (2018)
23. Tang, J., Qu, M., Mei, Q.: PTE: predictive text embedding through large-scale heterogeneous text networks. In: Proceedings of 21st International Conference on Knowledge Discovery and Data Mining, pp. 1165–1174 (2015)

24. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: large-scale information network embedding. In: Proceedings of 24th International Conference on World Wide Web (WWW), pp. 1067–1077 (2015)
25. Yang, C., Xiao, Y., Zhang, Y., Sun, Y., Han, J.: Heterogeneous network representation learning: survey, benchmark, evaluation, and beyond. arXiv preprint arXiv:2004.00216 (2020)
26. Lu, Y., Shi, C., Hu, L., Liu, Z.: Relation structure-aware heterogeneous information network embedding. In: Proceedings of 33rd Conference on Artificial Intelligence (AAAI) (2019)
27. Zafarani, R., Liu, H.: Social computing data repository at ASU (2009)
28. Zhang, C., Swami, A., Chawla, N.V.: SHNE: representation learning for semantic-associated heterogeneous networks. In: Proceedings of 12th International Conference on Web Search and Data Mining (WSDM), pp. 690–698 (2019)
29. Zhang, D., Yin, J., Zhu, X., Zhang, C.: MetaGraph2Vec: complex semantic path augmented heterogeneous network embedding. In: Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 196–208 (2018)