Applied Intelligence (2023) 53:19302–19316
https://doi.org/10.1007/s10489-023-04520-9

A similar structural and semantic integrated method for RDF entity embedding

Duong Thi Thu Van[1] · Young-Koo Lee[1]

Accepted: 9 February 2023 / Published online: 2 March 2023

Abstract
Resource Description Framework (RDF) graphs have become an important data source for many knowledge discovery algorithms and data mining tasks. However, most complex analyses that use knowledge discovery algorithms require data in a vector representation format. As a result, several RDF entity embedding techniques have emerged in which entities in the RDF graph are represented as low-dimensional vectors. These techniques generate sequences of entities using graph walk and use language modeling techniques to extract the feature from the sequences used to learn the embedding. However, sequences produced by graph walks only capture structural context; they are unable to capture latent context, such as semantically related information, which is an important property of RDF data. In this paper, we present a novel method that consists of a series of steps that generate sequences. These sequences not only capture structural context but also semantic property. The method for structural context includes (1) a new concept of similar entities in which tradeoffs are made between similar outgoing edges and outgoing nodes and (2) a new structural similarity, which calculates the similarity between two entities in each sequence. We can generate sequences based on structural similarity so that similar entities contain sequences with similar structures. The method for the semantic property combines sequences with the same semantic to generate latent sequences that cannot be generated by traversing the graph. This paper presents experimental results and a case study using real graphs to show that the proposed method outperforms existing methods in terms of quality and efficiency.

Keywords RDF graph · Graph embeddings

1 Introduction

Graphs are useful for depicting and modeling real-world entities and the relationships between these entities. Graphs are used in a variety of domains, including social networks, biological networks, the semantic web, and citation networks [1]. For example, graph modeling of resource description framework (RDF) datasets is used in the semantic web domain, with nodes representing subjects or objects and edges representing predicates between subjects and objects. However, due to the increasing size and complex structure of these graphs, users may find it difficult to explore and understand large graph data. Graph embedding approaches occur to reduce the cost of analytics by representing graph nodes as vectors in a vector space such that the relevant network properties are preserved. These vectors can be conveniently used to address various downstream graph analytic tasks [2–4], such as node classification, link prediction, community detection, or visualization.

To learn vector representations for the graph, several embedding techniques adopted the neural language model (NLM) [5, 6] by generating the graph data into sequences of nodes, which can be considered as sentences. NLMs then take those sentences as the input and represent each node in the graph as a vector. Different techniques used different strategies to generate sequences. DeepWalk [7] uses a truncated random walk, while node2vec [8] uses a biased second-order random walk to generate sequences. Although these methods work efficiently for node embedding, they are only applicable for normal graphs, which are undirected or have no edge label. To embed the entities of an RDF

✉ Young-Koo Lee
yklee@khu.ac.kr

Duong Thi Thu Van
thuvan0301@khu.ac.kr

1 Department of Computer Science and Engineering, Kyung Hee University (GlobalCampus), Street, Yongin, 1732, Geonggi, South Korea

graph, additional work is proposed [9–12]. RDF2Vec [9] is an extension of DeepWalk [7] which produces sequences of entities in RDF graphs. It uses BFS random walk and subtree graph kernel extracted from the target entity to generate the sequences. A biased approach [10] compared twelve different methods for weighting edges that result in different sequences of entities. Like many techniques on the normal graph, these use word2vec [5] as the modeling technique with input sequences following the pattern of $subject \rightarrow predicate \rightarrow object \rightarrow predicate \rightarrow object$.

Although the above existing techniques provide several strategies to generate sequences, they do not mention the feature of similar nodes which is an important point when converting the node to a vector. Xu et al. [13] suggested that selecting a proper node similarity measure is particularly important for finding effective node context (or neighbors). For one node, its context is all sequences that contain that node. Similar to NLMs, we aim to output very close vectors for similar nodes. Therefore, the sequence generation process should generate very similar contexts to very similar nodes. The basic idea behind graph embedding is to optimize low-dimensional embeddings to approximate node similarities in the original graph generated. As a result, the goal of this paper is to generate a set of sequences with similar structures for similar nodes. First, we propose a novel concept where similar entities trade-offs between the number of similar outgoing edges and outgoing nodes. By using entity similarity, we developed a structural similarity metric that calculates the similarity between two nodes in a sequence at the current time. This definition helps our method to generate similar structural sequences for similar entities.

Moreover, the graph embedding process will learn statistics based on the number of times each neighborhood pair appears. Each node has a huge diversity in the neighborhood definitions. However, recent works like [7, 9, 14] only use outgoing neighbors to generate sequences of a node. Therefore, they are unable to capture latent context, such as semantically related information, which is an important property of RDF data. On the expressive side, the meaning of latent dependencies among nodes should be captured. Our paper provides a method for combining sequences with the same semantics to generate latent sequences that cannot be generated by traversing the graph.

Furthermore, because RDF graphs contain literal values, applying graph embeddings to them is more difficult. Literal types in an RDF dataset include string, number, and date. These literal types have many different values, thus making it a difficult task to identify them all. As a result, previous work often ignored the literals in the dataset before building an RDF graph from the RDF dataset [9]. Some datasets, however, contain important literals [15, 16]. In this paper, we add an additional step to reprocess the data before generating the sequences. This step will process literal values to nodes in a graph by assigning the same identity to textual literal values with similar meanings.

We summarize the main contributions of this paper as follows:

1. We propose an approach to assigning a new identity to important literal values such that nodes with similar meanings are assigned the same identity.
2. A similar structure walk (simStructWalk) method generates the sequences for the nodes in the graph followed by the structural similarity.
3. Latent sequences generation. We propose a method called latent walk (LW) to generate latent sequences which are unable to be generated using graph walk, by combining two sequences based on given features.
4. Experiments with several benchmark datasets to evaluate our methods. We generate a corpus of sentences for each dataset using different techniques, and trained the embedding using two different types of language modeling: Skip-gram and bag-of-word. Classification and regression tasks are used to evaluate the quality of our embeddings to other methods. The first method is the random method (RW), which uses a random walk to create sequences. The second method is Rdf2vec which creates sequences using Weisfeiler–Lehman Subtree RDFgraph kernels. The results demonstrate that our methods improve the quality of the generated sequences.

The rest of this paper is organized as follows: Section 2 discusses related work. Background information is provided in Section 3. Section 4 presents the proposed method and implementation. Section 5 demonstrates the experimental results. Finally, Section 6 presents the paper's conclusions.

2 Related work

Currently, a large quantity of structured and unstructured data is being generated. Knowledge generation using these data needs a good representation of each word. Many methods have been developed to represent a word as a continuous vector such as word2vec [17], Glove [6], and Bert [18]. By adopting those language models for graph embeddings, many approaches take advantage of the word order in text documents, explicitly modeling the assumption that closer words in a sequence are statistically more dependent.

Neural language models Because of advances in machine learning, neural network approaches such as word2vec

[17] and Glove [6], have gained popularity. In [19], a popular feed-forward neural network-based architecture is proposed for estimating the neural network language model (NNLM). In this network, the learning for representing a word vector is done through the use of a linear projection and a non-linear hidden layer. In [20] and [21], a fast NNLM architecture is proposed which uses a single neural network hidden layer, providing faster learning of the word vector by the model.

The neural network-based approach The neural network-based approach provides better performance than the Latent Semantic Analysis and, for large data, is less computationally expensive than the Latent Dirichlet Allocation (LDA). In [5], a neural network-based approach for distributed word representation is proposed. The author proposed the use of two unsupervised word2vec models, a CBOW and a continuous-Skip-gram model. Subsequently, several neural network approaches have been proposed for graph embedding [7, 8]. DeepWalk [7] is one of the first approaches to learn embeddings by implementing methods designed to work on text [22]. DeepWalk uses Skip-gram as a tool to learn a model from a network. In particular, DeepWalk generates a context of nodes (sequences) through the use of a truncated random walk. Node2vec [8] is an extension of DeepWalk since it generates sequences through the use of a biased second-order random walk. Subgraph2vec [14] is another approach for learning embeddings for rooted subgraphs. Unlike previous techniques, Subgraph2vec does not use random walks to generate context. Instead, in Subgraph2vec, the context of a node is simply defined by its neighbors. Subgraph2vec also recognizes structural equivalence by embedding nodes with the same local structure at the same point in space.

Translation-based models TransE [23] (Translation Embeddings) is one of the most successful of these methods. TransE constructs entity and relation embeddings according to the translation from the head entity to the tail entity. This method assumes that the relations of some words in the embedding space can be calculated by the difference between their vectors. However, this approach has the limitation of not being able to deal with one-to-many, many-to-many, many-to-one, and reflexive situations. The TransH model solved these issues by including a hyperplane with the translation operation [24]. TransH is divided into two vectors, namely, the norm vector of the hyperplane, and the translation vector in a hyperplane. The TransR model takes both TransE as well as TransH models to embed the relations among the entities in the equivalent semantic space [25]. This method combines the entity and relation embedding into a multiple relation space as well as into a separate entity space that allows the modeling of entities from numerous aspects.

Literal-based models Ristoski and Paulheim [9], Cappuzzo et al. [26] approaches removed literals in embeddings, thus leading to poor predictions. In [15], the author discusses how literals can improve learning by creating learning embeddings for entities, which have no link with any entity in the KG, and by enhancing the entity representation for structure-based embedding models. In [16], the author introduces LiteralE, a generalized method where numeric literal information is included to support any latent feature method. This method enriches embeddings with literal information that improves the prediction rate. MADLINK [27] outperformed in link prediction task by adapting the seq2seq [28] encoder-decoder architecture with attention layer to obtain a cumulative representation of the paths extracted for each entity from the KG. This work also showed that the textual description of the entities in the KGs contains rich semantic information and the graph structure provides the contextual information of the entity from the neighboring nodes. Therefore, this paper uses SBERT [29] to extract the latent representations from entity descriptions. In this paper, we process literal values to be vertices in a graph by assigning the same identity to the textual literal values which have similar meanings by adapting doc2vec and KNN.

Language modeling based RDF graph embedding A language modeling approach has been adopted to represent the RDF graph entities in [9, 10], and [26]. RDF2vec [9] is an extension of DeepWalk [7] for embedding the RDF subgraph using a BFS random walk. Using graph walks and Weisfeiler-Lehman Subtree RDF graph kernels, a set of sequences is generated from the local information in a graph. Next, a neural language is used to train the generated sequences, which are used to estimate the likelihood entities. Finally, a graph entity is represented as a vector of the latent numerical feature. The vector generated from each entity shows that they are closely related to each other. Cochez et al. [10] is a biased version of RDF2vec [9], which uses predicate and object frequency to calculate the probability of the following edges. Guan et al. [12] proposed an embedding model with concepts that jointly into a semantic space.

However, rdf2vec [9] or its extension does not claim that similar entities will have similar sequences. In this paper, first, we provide the set of sequences that demonstrate similar entities with similar structure sequences. Moreover, recent works like [9] or [26] use random walks that use outgoing neighbors to generate sequences of a node. Our paper provides a method for combining sequences with the same semantics to generate latent sequences that cannot be generated by traversing the graph. Second, to generate latent sequences, we combine sequences with the same semantics that cannot be generated by graph walk. Third, we do not remove literal values from the RDF dataset. We use an additional step to preprocess the dataset, which turns the literal values into valuable values. The paths provided by the second and third steps are not included in the rdf2vec method.

Riccardo et al. [30] proposed an approach to obtain vector embedding for entities in a relational database. From a very high point of view, similar to our method as well as other existing graph embedding techniques, they first generate a corpus of sentences from the original structured data, and then apply a neural language model to the text corpus to obtain a vector embedding model. To generate the sequences, they first represent the relational data into a graph form, based on a tripartite Heterogeneous Graph data structure. They then use random walks to travel the constructed graph representation of the relational data and represent the walks as sentences. The basic difference between this and our method is the type of data that each method applies to. While Riccardo et al.'s method is applicable to relational databases, our method is applicable to RDF data.

3 Preliminaries and problem statement

Given an RDF graph, we first convert it into a set of entity sequences, which we then use to train a neural language model so that similar entities have vectors that are close in the vector space.

3.1 Graph-based RDF data model

Definition 1 RDF Graph. An RDF graph $G = (V, E)$ is a simple directed graph where V and E are the set of vertices and edges, respectively. A directed edge from u to v is denoted as (u, v, p), where u and v are the vertex IDs of that edge and p is the label ID of the edge. The vertex IDs are the identifications of the Subject, Object, and Predicate.

In an RDF graph, vertex ids are the identification of URIs. An object, however, can also be a blank or literal value. Because literal values are not URIs, they are difficult to identify. Literals in an RDF dataset can be of string or number types. Almost all existing methods ignore the literal values. However, some kinds of literal values contain important information [15, 16]. In computer science, for example, a DBLP dataset consists of bibliography data. Each paper may cite or be cited by other papers, resulting in the formation of a citation network. Each paper in this dataset has its title as a literal string representation. Attention to the title of the paper is important when searching for specific words or grouping papers by content because the title is a summary of the content. Hence, significant literal values should be employed. We present a preprocessing step in this paper to assign an identity to literal values. The details of this step are in Section 4.1.

3.2 Sequences generating problem

Definition 2 Sequence depth d of node v. Given an RDF graph $G = (V, E)$ and a depth d, sequences S_v of each node $v \in V$ are the paths of several connected edges starting from node v, where each edge (v, u, p) creates a sequence depth 3.

Definition 3 Random Walk on an RDF graph Given an RDF graph $G = (V, E)$, the random walk provides a walk that starts from a node $v \in V$ and walks to any node that is outgoing of node v.

The graph walk can produce all of the possible sequences. For each node $v \in V$, the graph walk will generate a set of sequences S such that the first word is v and the following word is an outgoing edge and a corresponding neighbor of v. However, using a graph walk will yield a large number of sequences. Using all of these sequences for model training is impractical. In reality, we only use their bias in all possible sequences. A random walk is one possible solution to the above-mentioned problem. However, good sequences can be missed. For finding good sequences we should select a proper node similarity measure that is particularly important in graph [13]. Based on the node similarity measures, we generate similar contexts for similar nodes.

Therefore, we adapt the random walk by including conditions such that the more similar nodes, the more similar the set of sequences generated. We propose the structural similarity metric, which can direct the walk to travel along the priority path. The details of this step will be provided in Section 4.2.

Graph walk techniques use outgoing neighbours to generate sequences, which capture only structural context [7, 9, 14]. They are unable to capture latent context, such as semantically related information, which is an important property of RDF data. Therefore, we add one more step after structural context generation. This is the step in which latent sequences are generated to provide a new set context that connects related semantic sequences. Section 4.3 shows how this is done.

3.3 Neural language models

3.3.1 Word2vec

Word2vec is a neural network approach that converts each word in a text corpus to a corresponding vector such that similar words are close to each other in the vector space. It can capture the context of a word in a document with a semantic and syntactic similarity, and its relation to other words. It uses two models (both involving Neural Networks): Skip-Gram and Continuous Bag Of Words (CBOW).

CBOW [5] is a well-known unsupervised word2vec model among researchers. The CBOW model is used to predict a target word by utilizing context words. Let us use the sentence, "birds are flying over the sea" to demonstrate this method. The sentence can be thought of as a pair of (context, target). Here context window size can be variable. If the size is 2, we can make CBOW as ([flying, sea], birds), ([the, sea], birds), ([over, flying], sea). Based on context, CBOW predicts the target. Figure 1 depicts the CBOW architecture. The complexity of training is calculated as $Q = NXD + DXlog_2(V)$, where N is the dimension of the input layer, D is the dimension of the project layer, and V is the size of the vocabulary. Given the size of the context window b, the objective of the CBOW model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\log p\left(w_t \mid w_{t-b}...w_{t+b}\right) \tag{1}$$

Another popular unsupervised model is Skip-Gram [5] which predicts the most relevant words (i.e., context) based on a target word. This model is a reverse approach to the CBOW model. In Skip-gram, words are statistically more related and also statistically dependent on each other. The architecture of Skip-gram is depicted in Fig. 1. The CBOW model predicts a single target word according to multiple inputs that maximizes the log-likelihood probability distribution. However, in the Skip-gram model, a target word predicts the context word. For instance, (birds, flying), (Birds, sea), (flying, sea). Skip-gram provides a more accurate output than CBOW. However, due to increased computational needs, Skip-gram is slower than CBOW. The objective function of Skip-gram is represented by (2):

$$\eta = \sum_{t=1}^{T}\log P\left(w_{t-b} : w_{t+b} \mid w_t\right) \tag{2}$$

In (2), b is the window size (context width), and the sequence of words is represented by $w_{t-b} : w_{t+b}$, where the word w_t itself is eliminated from this sequence. The probability $P\left(w_{t-b} : w_{t+b} \mid w_t\right)$ is calculated using (3)

$$\prod_{-b\leq j\leq b,\, j\neq 0} P\left(w_{t+j} \mid w_j\right) \tag{3}$$

where the given word w_t is independent of the contextual words $w_{t+j} : w_j$. The computation for $P\left(w_{t+j} \mid w_j\right)$ is as follows

$$P\left(w_{t+j} \mid w_t\right) = \frac{\exp\left(v_{wt}^T\right)\left(v'_{t+j}\right)}{\sum_{w=1}^{W} exp(v_{wt}^T)(v'_w)} \tag{4}$$

where the input and output vectors are v_w and v'_w of w, respectively.

Figure 1 shows the architecture of CBOW and Skip-gram.

3.3.2 Doc2vec

Doc2vec [31] is an unsupervised NLP algorithm that generates vectors from sentences or paragraphs. This idea originated from word2vec where a vector is generated from a word. Based on word2vec, the Distributer Memory version of Paragraph Vector (PV-DM) is a doc2vec method that works like a memory to remember the missing context

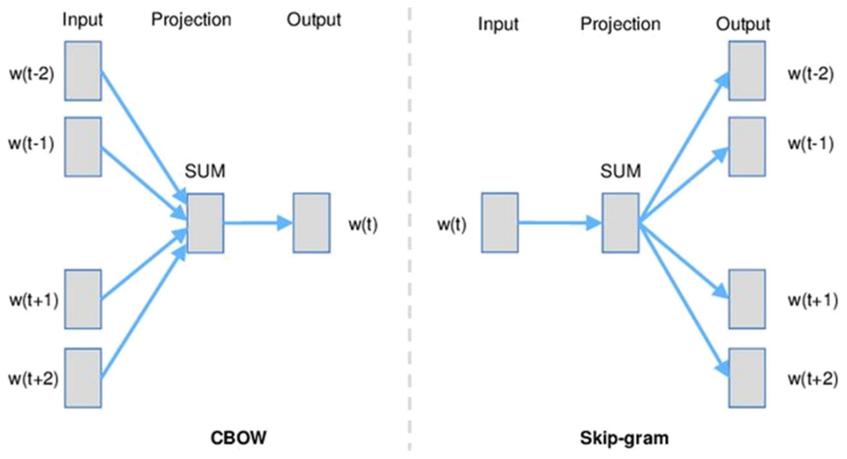


Fig. 1 CBOW and Skip-gram architecture

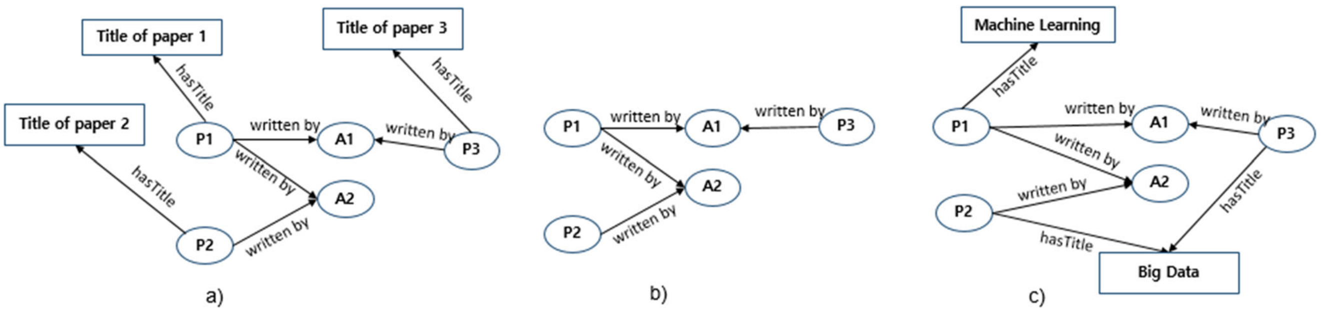


Fig. 2 An example of the DBLP dataset. Figure a) is an RDF graph G_1, b) the RDF graph that ignores literals G_2, and c) an RDF graph that identifies the titles of paper, G_3

or topic from a paragraph. Another popular method is the Skip-gram-based Distributed Bag-of-Word Version of Paragraph Vector (PV-DBOW). PV-DBOW is faster than PV-DM and requires less memory. Doc2vec is particularly effective in binary classification (e.g., positive and negative sentiments) [32].

4 Our proposed method

Our proposed method involves a network that is represented by an RDF graph $G = (V, E)$. The goal of this method is to extract node embeddings in such a way that similar nodes are mapped to points close to each other in the vector space. To get the node embedding of the graph, the first step is to generate sequences of entities from a graph, which captures the surrounding knowledge of each entity. Next, a neural language modeling technique is used to embed entities into a vector space. As discussed previously, the effectiveness of vector representation depends on the sequences in use and the objective. Therefore, in this section, we introduce three methods in turn to improve the quality of generated sequences.

4.1 Assigning a new identity to a literal

We already discussed the significance of literal values in the RDF dataset, particularly the textual literal that shows a subject's description, in Section 3. The most important question is how to use them effectively. The issue is that the RDF dataset contains many textual literals. If we treat each distinct textual literal as a separate node in the graph, a large number of nodes will be added, and each subject will be connected to a different literal. As a result, it shows no relationship between subjects that have similar literal values. Our work aims to use literal value similarity to connect different subjects through similar literal values. Instead of removing all literal values from the dataset before building the RDF graph or representing each textual literal by a single node, we assign the same identity to the textual literal values which have a similar meaning.

An example is shown in Figs. 2 and 3 which illustrate the difference between using and ignoring literals in an embedding problem. We begin with the small example shown in Fig. 2 and which generates the results shown in Fig. 3. In this example, the title of papers are literals of type String, the original RDF graph is shown in Fig. 2a),

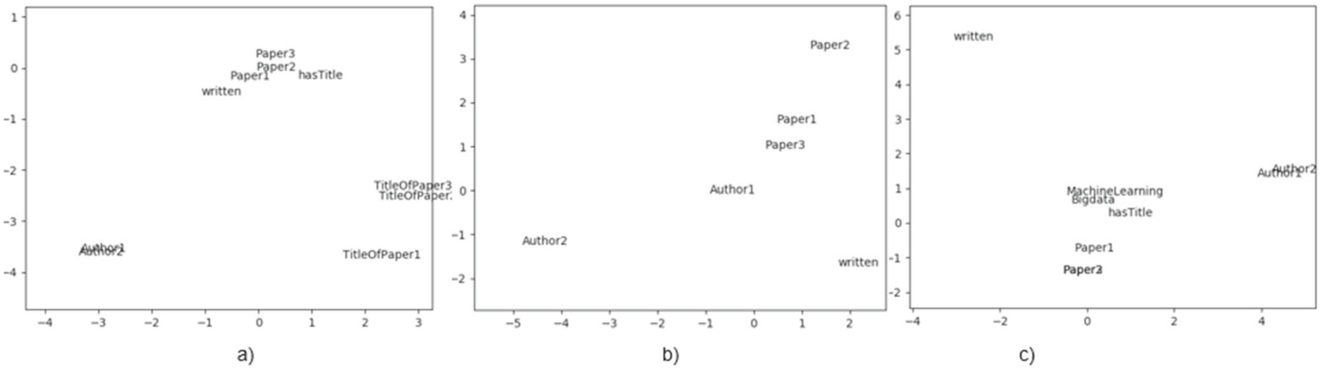


Fig. 3 a), b), and c) are embeddings of data from Fig. 2a, b, and c, respectively

and the graph after ignoring the literal is shown in Fig. 2b). The graph in Fig. 2a) provides more information than the graph in Fig. 2b). This is especially true when the title is the relevant node that decides when two papers are similar. If we remove the title value because it is textual literal, we also remove the predicate corresponding to them (Fig. 2b)). Clearly, every paper contains the node "hastitle" that makes every paper the same when we calculate the probability of Paper and "hastitle".

Therefore, in this paper, we add an additional step to process the data before generating the sequences. For literal values that are textual, we try to understand their meaning by converting them to vectors. Similar literals will be assigned the same identity. Clearly, the number of identities is less than the number of literals. Therefore, important literal values are selected and retained to train the model. The example shown in Fig. 2c) illustrates our proposed method. We group similar titles into the same group. Figure 3 shows the visualization of paper 1 (P1), paper 2 (P2), and paper 3 (P3) in vector spaces of graphs a, b, and c respectively. The graphs in Fig. 3a) and b) do not have differences to identify which papers are highly similar to the others. However, Fig. 3c) shows that paper 2 is more similar to paper 3 than to paper 1. According to the semantics of the dataset, the embeddings shown in Fig. 3c) show the most suitable results.

4.2 Sequence generation by similar structure walk (simStructWalk)

Random walk techniques can either generate all possible sequences or miss some sequences. In some cases, missing sequences are significant for the training model. The significant sequences contain a set of ordered words that more frequently appear in the context. Consider the illustration in Fig. 4, where node $A1$ is similar to node $A2$ because their neighborhoods are similar. The two red paths provide useful sequences for training the model because they have similar structures. Nevertheless, not all sequences are always generated. In this example, we assume that the

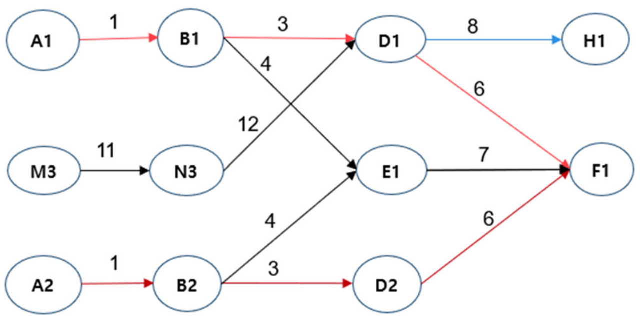


Fig. 4 An example RDF graph

path $A1 \rightarrow 1 \rightarrow B1 \rightarrow 3 \rightarrow D1 \rightarrow 8 \rightarrow H1$ is generated instead of red path $A1 \rightarrow 1 \rightarrow B1 \rightarrow 3 \rightarrow D1 \rightarrow 6 \rightarrow F1$. Consider structure of two red paths $A1 \rightarrow 1 \rightarrow B1 \rightarrow 3 \rightarrow D1 \rightarrow 6 \rightarrow F1$ and $A2 \rightarrow 1 \rightarrow B2 \rightarrow 3 \rightarrow D2 \rightarrow 6 \rightarrow F1$ they have same structure $AnyNode \rightarrow 1 \rightarrow AnyNode \rightarrow 3 \rightarrow AnyNode \rightarrow 6 \rightarrow F1$. The frequency of this structure is larger than the structure of the blue path $A1 \rightarrow 1 \rightarrow B1 \rightarrow 3 \rightarrow D1 \rightarrow 8 \rightarrow H1$, which means two red paths show more similar structure than the blue one. Thus, in a case where we cannot generate all of the sequences, we prioritize generating the red paths first. To do that, first, we define entity similarity, which will guide the walk to follow other similar entities. Second, a similar structural walk to priority walk to the same structural paths of similar nodes to generate as many significant sequences as possible.

To generate sequences that have similar structure paths, we define *similar nodes* and the *structural similarity* of two nodes. We utilize the property of nodes in the RDF graph that are subjects or objects and connected by the predicate. Two nodes having a similar set of predicates mean they have the same schema. Additionally, if they have a similar set of neighborhoods, they are similar to each other.

Definition 4 Entity similarity. The similarity of two vertices u and v is calculated by the sum of the ratio of similar connections (predicates) to total connections and the ratio of similar neighborhoods to total neighborhoods. Let $s(u, v)$ denote the entity similarity of two vertices u and v. The formula is displayed in (5).

$$s(u, v) = w\frac{\mid predicate(u, v) \mid}{\mid totalPredicate(u, v) \mid} + (1-w)\frac{\mid neighbor(u, v) \mid}{\mid totalNeighbor(u, v \mid} \tag{5}$$

where $\mid predicate(u, v) \mid$ presents the number of the intersect of the set of edge label ids(predicate ids) of u and v. $\mid neighbor(u, v) \mid$ is the number of the intersect of neighbors of u and v. $\mid totalPredicate(u, v) \mid$ is the number of the union of edge label ids of the edges of u and v. $\mid totalNeighbor(u, v) \mid$ is the number the union of edges of u and v. Finally, w is a weight to adjust the priority where $w \in (0, 1)$. Our method considers all incoming and outgoing edges of the node.

Definition 5 Structural similarity. The structural similarity of two nodes u and v, given a previous node a, is an average of the similarity of a and the incoming of node v.

Let $f(u, v)\{a\}$ denote the structural similarity distance between u and v, where a is a previous vertex of u. The formula for calculating $f(u, v)\{a\}$ is shown in (6).

$$f(u, v)\{a\} = \frac{\sum_i^n s(a, b_i)}{n} + s(u, v) \tag{6}$$

where b_i is the incoming node of v.

Let $sc(u, a)$ represent the set of nodes that are similar to u, given previous node a. $sc(u, a)$ contains the nodes that have an f value greater than or equal to a threshold δ (i.e., $f \geq \delta$).

Definition 6 Weight of next node. The weight of the next node q of the node p in the current path is defined as a fraction of the number of possible walks from p to q and the total possible walks from p.

$$w(p, q) = \frac{M(p \mid q)}{\mid totalNeighbor(p) \mid + \mid totalPredicate(p) \mid} \tag{7}$$

$$M(p \mid q) = \mid edge(sc(p, a), predicate(p, q), any) \mid + \mid edge(sc(p, a), any, q) \mid \tag{8}$$

where $M(p \mid q)$ represents the total cases where q can be walked from p or from the nodes which are similar to p. $\mid edge(sc(p, a), predicate(p, q), any) \mid$ means that any node in $sc(p, a)$, the label of an edge must be $predicate(p, q)$, and go to any other nodes. On the contrary, $\mid edge(sc(p, a), any, q) \mid$ means any edge that comes from a node in $sc(p, a)$ to node q passes through any predicate. Although we select the following node at random from the list of candidates, we give higher priority to those nodes with higher weight. As a result, nodes with higher weight will have a greater chance of being walked through.

The overall idea of generating sequences (paths) is that choosing the next node is based on the previous nodes of the current sequence. To decide the next node in a sequence, we calculate the weight that represents the number of similar paths using that node. Figure 5 depicts the overview of our similar structural walk. For each node u in graph G, we do the following steps to generate one sequence:

- The first step (step 1 in Fig. 5) is to obtain $sc(u, a)$, the set of similar nodes of node u based on the current path.

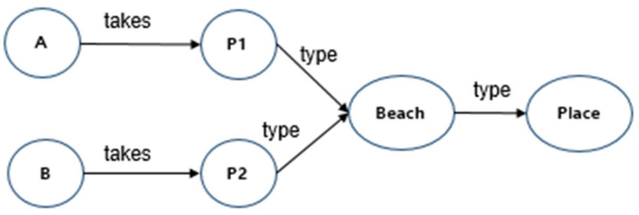


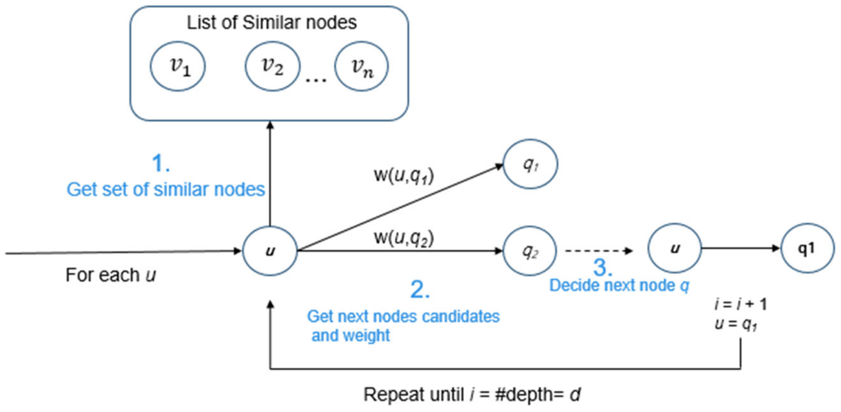
Fig. 6 A graph example for latent sequences

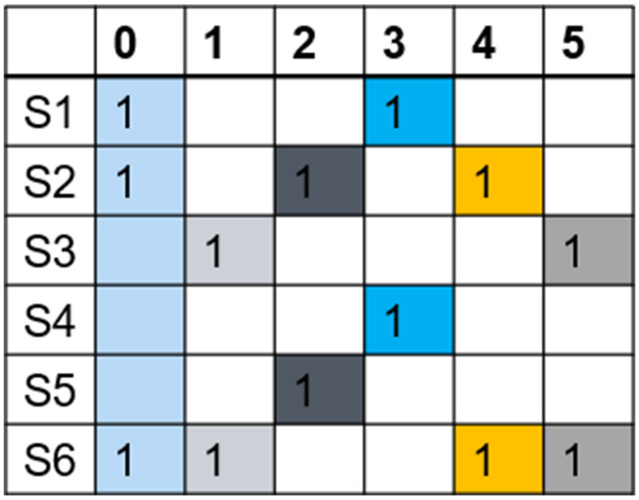
- The second step (step 2. in Fig. 5) is to collect the next node candidates and their corresponding weights. We use (7) to calculate the weight of each candidate.
- The third step (step 3. in Fig. 5) is to select the next node q from the list of candidates such that the node with a higher weight has a higher chance of being selected. The selected node q becomes node u for the next iteration.
- Steps 1, 2, and 3 are repeated until the path contains enough d nodes.

4.3 Latent sequences generation

Unlike normal graph problems, RDF nodes are called similar when they have the same semantics. We consider the example shown in Fig. 6. Given depth 4, we can generate two sequences that are $A \rightarrow takes \rightarrow P1 \rightarrow type \rightarrow Beach$ and $B \rightarrow takes \rightarrow P2 \rightarrow type \rightarrow Beach$. The semantics of the two sequences indicates that A and B take the same type of photo. Thus, the sequences should indicate that $A \rightarrow and \rightarrow B \rightarrow took \rightarrow photo \rightarrow type \rightarrow beach$ or $Beach \rightarrow is \rightarrow taken \rightarrow by \rightarrow A \rightarrow and \rightarrow B$. For a window size of three, these two sequences estimate a higher probability that A will occurs with B than other sequences. Both Continuous Bag of Words (CBOW) and Skip-gram methods use the joint probability of A and B occurring so that the latent sequences are good sequences for training purposes. These are called latent sequences because they cannot be generated by traveling the graph.

Fig. 5 Our similar structural walk architecture





	0	1	2	3	4	5
S1	1			1		
S2	1		1		1	
S3		1				1
S4				1		
S5			1			
S6	1	1			1	1

Fig. 7 A matrix of sequences

Definition 7 (Latent Sequences) Given two sequences s and s' having the same depth d, and a feature F where $s = v_1 \rightarrow p_2 \rightarrow v_3 \rightarrow ... \rightarrow p_{d-1} \rightarrow v_d$, and $s' = v'_1 \rightarrow p'_2 \rightarrow v'_3 \rightarrow ... \rightarrow p'_{d-1} \rightarrow v'_d$, latent sequence created by s and s' is a sequence that is merged by s and s' if s and s' contain any similar nodes in F.

There are many sets of new triples that can be created by using an RDF schema. However, checking and querying the data through the RDF schema will take a lot of time. Moreover, many datasets lack their schema. Therefore, we aim to generate new sequences without requiring the schema. Instead of using schema, we provide the method to create new sequences based on the properties/features. Given one or more features $F = \{f_1, f_2, ..., f_n\}$, that are used as a feature selection. In other words, given a list of priority properties of entities, we generate sequences that use feature selection to connect the nodes.

We create a matrix $M(m, n)$ (ex, Fig. 7), where m is the number of sequences and n is the number of entities that belong to feature F. In this matrix, value 1 indicates that the sequence at this row contains corresponding entities. If two rows have value 1 in the same column, they can be merged into a single sequence.

5 Experiment

We evaluate our approach on two different tasks that are classification and regression. Our experiment is divided into three steps: The first step is to convert the RDF graphs into a set of sequences. In the second step, we use the set of sequences generated in the first step to build embedding models. The last step is to evaluate the accuracy of classification and regression tasks through a well-known set of machine learning tasks. To show the efficiency of our proposed methods, we compared the accuracy of classification and regression tasks to other methods. The first method is known as the random method (RW) and it uses a random walk to create sequences. The second method is known as rdf2vec and it creates sequences using Weisfeiler–Lehman Subtree RDFgraph kernels (we used the rdf2vec experimental results from publication [9]). In the second step, to get the embeddings, we also follow the rdf2vec's setup to use the same CBOW and Skip-Gram models. For the classification task, we used Naive Bayes, k-Nearest Neighbors (k = 3), C4.5 decision tree, and Support Vector Machines. We used k-Nearest Neighbors, linear regression, and regression tree for regression. For comparison to rdf2vec results, we set up the SVM classifier to optimize the parameter C in the range 10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3 similar to rdf2vec's setting.

- RW designates the method that only uses a random walk.
- simStructWalk designates the method that only uses a similar structural walk.
- LW designates the method that uses Latent walk
- RW + LW designates a random walk combined with a latent walk.
- simStructWalk + LW designates a similar structural walk combined with a latent walk.
- Rdf2vec uses Weisfeiler-Lehman Subtree RDFgraph kernels for the sequence generation process.
- Rdf2vec + LW designates rdf2vec combined with a latent walk for the sequence generation process.

We implemented a sequence generation program in Scala using GraphX [33]. The training model is developed in python. Experiments are conducted on an Intel(R) Core(TM) i5-10400 CPU running Windows 10 with 16.00GB of RAM. For evaluating the performance of our RDF embeddings in machine learning tasks, we evaluate on a set of benchmark datasets. We used the following five real datasets: AIFB, MUTAG, DBLP,[1] CiteSeer-M10,[2] and DBpedia [34]. The summary of these datasets is shown in Table 1 with the total number of nodes, the total number of triples, the total number of target nodes that show the number of entities used for learning, the number of classes for the classification task, and the last column is rating, that shows the dataset's ground truth contains rating value or not. Details of these datasets are included in the section in which they are used. The source code of this implementation is available at https://github.com/vandtt/RDFEntity2Vec. This source code was modified from the source code Trip Walk.[3]

[1] http://arnetminer.org/citation (V4 version is used)

[2] http://citeseerx.ist.psu.edu

[3] https://github.com/chrisPiemonte/TripWalk

Table 1 Dataset properties

Dataset	#Nodes	#Triples	#Target Nodes	#Classes	Rating
AIFB	8,291	29,226	176	4	No
MUTAG	22,549	2,247931	340	2	No
DBpedia- City	>3 M in total	>400M in total	212	3	Yes
DBpedia- Movie			2,000	2	Yes
DBpedia- Album			1,600	2	Yes
DBpedia-AU UUniversity			960	3	Yes
DBLP	60,744	52,890	60,744	3	No
CiteSeer-M10	10,310	77,218	10,310	9	No

5.1 Evaluation on different kinds of walks

In this section, we will show the efficiency of our similar structural walk and latent walk compared to a random walk and rdf2vec walk on two tasks that are classification and regression tasks.

5.1.1 Datasets

We will use three out of five real-world datasets that we mentioned in the previous section [35] in this section.

The first is the AIFB dataset that describes staff, research groups, and publications from the AIFB research institute. AIFB contains 8,291 triples and 29,226 nodes in total. The ground truth file for the classification task consists of 178 members of a research group which is divided into 4 classes.

The second is the MUTAG dataset which is distributed as an example dataset for the DL-Learner toolkit (http://dl-learner.org). It contains information about complex molecules that are potentially carcinogenic and mutagenic (about 340 substances). The third is the cross-domain RDF dataset which the DBpedia [34] project builds. This is a large-scale, multilingual knowledge base created by extracting structured data from Wikipedia. The DBpedia project extracts knowledge from 111 different language editions of Wikipedia. DBpedia can be used as Linked Data on the Web since it covers a wide variety of topics and sets RDF links pointing toward various external data sources. Thus, DBpedia has developed into a central interlinking hub in the Web of Linked Data and is a key factor for the success of the Linked Open Data initiative. In this experiment, we used the English version of the 2015-10 DBpedia dataset. We evaluated this dataset based on four main features such as city, movie, album, and university. This dataset also contains the ground truth files for each feature, rating values for the regression tasks, and class labels for the classification tasks. The rating value is the ratio between the positive and the total number of feedback provided by users. Based on the rating value, the subject is assigned to the corresponding class label to show the summary of feedback.

The Mercer [36] dataset was used for evaluating the classification. This dataset contains quality of living measures for a list of cities. This dataset contains 212 cities and is divided into three classes of quality. A movie dataset is retrieved from Metacritic.com, which provides an average rating from reviews for a list of movies [37]. We used 2000 out of the 10,000 movies in this dataset for our classification task, which is divided into two classes "good" and "bad". Similar to the movies dataset, the Metacritic Albums dataset was retrieved from Metacritic.com. This dataset contains an average rating of reviews for a list of albums [38]. We chose 1,600 albums from this dataset for inclusion in this experiment. The American Association of University Professors (AAUP) dataset contains a list of universities and average salaries which were both set as target variables for classification. This dataset contains 960 entries and is divided into three classes.

5.1.2 Experimental results

For both the AIFB and MUTAG datasets, we generated a sequence depth of 5 with 4 walks per node. Our definition of depth differs from the definition of depth in rdf2vec. In rdf2vec, each path $v_1 \rightarrow p \rightarrow v_2$ is counted as 1, while our definition count this as 3. Therefore, a depth of 5 in our method is the same to a depth of 2 in the rdf2vec method. Tables 2 and 3 show classification results from the AIFB and MUTAG datasets, respectively. Especially, for AIFB and MUTAG dataset, we used sentences generated from the rdf2vec method as input to the latent walk. Then we measured the accuracy of the classification task. This improved results versus only using rdf2vec. For the AIFB dataset, the feature was set to $F = Type, Publication$. For the MUTAG dataset, the feature to generate latent sequences was set to $F = Type, carcinogenesis$. We apply both the CBOW and Skip-Gram models to the generated sequences with the same parameters as used in rdf2vec method. These include setting the window size to 5, the number of iterations to 10, using negative sampling for optimization, and setting negative samples to 25. In this experiment, we only

Table 2 Classification results on the AIFB dataset

		NB	KNN	SVN	C4.5
CBOW 500	RW	0.57	0.61	0.58	0.6
	Rdf2Vec	0.69	0.69	0.83	0.73
	simStructWalk	0.71	0.72	0.76	0.75
	RW + LW	0.83	0.84	0.88	0.82
	RDF2vec + LW	0.83	0.82	0.85	0.82
	simStructWalk + LW	0.85	0.83	**0.88**	0.83
SG 500	RW	0.70	0.72	0.61	0.71
	Rdf2vec	0.89	0.89	0.93	0.70
	simStructWalk	0.71	0.72	0.76	0.75
	RW + LW	0.83	0.84	0.88	0.82
	RDF2vec + LW	0.91	0.9	0.94	0.85
	simStructWalk + LW	0.88	0.93	**0.97**	0.87

Bold entries represent the best accuracy for each model

compare results for 500 dimension embeddings since 200 of the dimensions showed lower accuracy than the other 500 dimensions. Because AIFB and MUTAG datasets do not show the ground truth for the regression task, we can not show the experimental results for these datasets on regression.

These same settings were used for the AIFB and MUTAG datasets as well as for four specific features of the DBpedia dataset. For these three datasets, we generate sequences with depth 9 (depth of 4 for rdf2vec) with only 100 walks per node (note: RW and rdf2vec must use 500 walks per node). Our similar structural walk claims that with a smaller number of walks, the generated sequences are good enough for training. In this work, we focus to show the efficiency of using similar structural Walk and latent walk together. Thus, on average, using the latent walks achieves 30% more accuracy than the methods that do not use the latent walk method. For the regression task, we use root mean squared error (RMSE) to show the evaluation and the results are calculated using stratified 10-fold cross-validation. Tables 4, 5, 6, and 7 also show the RMSE of regression tasks. In most cases, the sequences generated by both similar structural walk and latent walk provide the best performance.

Table 3 Classification results on the MUTAG dataset.

		NB	KNN	SVN	C4.5
CBOW 500	RW	0.75	0.68	0.8	0.69
	Rdf2Vec	0.86	0.71	0.91	0.67
	simStructWalk	0.78	0.76	0.89	0.65
	RW + LW	0.88	0.85	0.90	0.74
	RDF2vec + LW	0.87	0.80	0.93	0.85
	simStructWalk + LW	0.89	0.88	**0.93**	0.75
SG 500	RW	0.73	0.71	0.76	0.72
	Rdf2vec	0.82	0.73	**0.96**	0.70
	simStructWalk	0.73	0.68	0.76	0.65
	RW + LW	0.85	0.82	0.87	0.77
	RDF2vec + LW	0.85	0.84	0.87	0.85
	simStructWalk + LW	0.83	0.85	0.92	0.79

Bold entries represent the best accuracy for each model

5.2 Benefit of assigning a new identity to literal nodes

In this part of the experiment, we demonstrate the efficiency of assigning a new identity to literals in the classification task. In this case, we combine the assignment of a new identity step with several different ways of generating sequences. We used two real datasets DBLP and CiteSeer-M10 which contain textual literal. We adopt a neural network Skip-gram to learn the vector representation of the document, particularly, Doc2vec. After representing literals by vectors, we use KNN to group the list of similar literals.

- Let NL+ RW denote the algorithm that is produced by combining the step assigning a new identity to literals with a random walk to generate sequences.
- Let NL + simStructWalk denotes the algorithm that is produced by combining the step which assigns a new identity to literals with our similar structural walk to generate sequences.
- let NL + simStructWalk + LS denote an algorithm that is produced by assigning a new identity to literals and then using both our similar structural walk and latent walk to generate sequences.

5.2.1 Dataset

We used two real datasets: 1) DBLP dataset which contains bibliography data for computer science. Each paper may cite or be cited by other papers, which naturally forms a citation network. Moreover, each paper is assigned to one class which was used for ground truthing for our experiment. The DBLP network consists of 60,744 papers (nodes) and 52,890 edges in total. The ground truth contains all papers and is divided into three classes.

2) CiteSeer-M10 is a subset of CiteSeerX data 2. This dataset contains information about scientific publications for 10 distinct research areas. This dataset consists of 9 multidisciplinary classes, 10,310 publications, and 77,218 total edges. Each paper belongs to one class and has one document to describe the paper. The edges in this dataset represent citation links between two papers. In this dataset, we used the class property for ground truth in our experiment. A total of 10,310 papers and nine classes are included in this ground truth file.

Table 4 Classification and Regression results on the DBpedia-City dataset

Model	Method	Classification				Regression		
		NB	KNN	SVN	C4.5	KNN	LR	M5
CBOW 500	RW	0.65	0.67	0.81	0.64	18.22	40.30	18.78
	Rdf2Vec	0.59	0.71	0.76	0.66	12.46	14.99	14.66
	simStructWalk	0.71	0.73	0.85	0.65	12.67	17.21	14.23
	RW + LW	0.77	0.76	0.86	0.72	13.80	30.07	17.79
	simStructWalk + LW	0.77	0.79	**0.89**	0.74	11.08	14.79	**11.02**
SG 500	RW	0.53	0.50	0.61	0.57	22.34	27.17	13.46
	Rdf2vec	0.58	0.72	0.76	0.67	13.25	14.73	16.80
	simStructWalk	0.70	0.69	0.73	0.62	13.68	16.90	15.60
	RW + LW	0.74	0.76	0.80	0.73	16.40	20.42	12.30
	simStructWalk + LW	0.81	**0.84**	0.77	0.74	**10.50**	13.50	11.57

Bold entries represent the best accuracy for each model

Table 5 Classification and Regression results on the DBpedia-Movie dataset

Model	Method	Classification				Regression		
		NB	KNN	SVN	C4.5	KNN	LR	M5
CBOW 500	RW	0.51	0.52	0.51	0.49	25.35	23.30	27.80
	Rdf2Vec	0.65	0.79	0.82	0.73	17.45	15.90	15.73
	simStructWalk	0.61	0.68	0.75	0.68	16.80	18.33	16.83
	RW + LW	0.72	0.77	0.88	0.78	20.50	18.20	17.35
	simStructWalk + LW	0.70	0.78	**0.91**	0.84	15.60	15.58	**13.40**
SG 500	RW	0.71	0.68	0.76	0.67	23.17	25.70	20.33
	Rdf2vec	0.65	0.80	0.83	0.72	17.14	15.66	15.67
	simStructWalk	0.73	0.74	0.79	0.76	15.20	24.40	13.50
	RW + LW	0.76	0.75	0.86	0.80	17.22	15.45	15.70
	simStructWalk + LW	0.83	0.84	**0.89**	0.87	15.15	17.60	**13.05**

Bold entries represent the best accuracy for each model

Table 6 Classification and Regression results on the DBpedia-Album dataset

Model	Method	Classification				Regression		
		NB	KNN	SVN	C4.5	KNN	LR	M5
CBOW 500	RW	0.54	0.51	0.52	0.53	23.15	13.20	13.75
	Rdf2Vec	0.69	0.71	0.75	0.65	12.60	11.49	11.48
	simStructWalk	0.62	0.63	0.71	0.69	14.70	11.35	11.90
	RW + LW	0.67	0.65	0.73	0.69	12.45	11.70	11.60
	simStructWalk + LW	0.73	0.75	**0.80**	0.78	12.90	**10.75**	10.85
SG 500	RW	0.58	0.57	0.63	0.56	22.40	13.58	11.40
	Rdf2vec	0.70	0.74	0.78	0.67	12.11	11.20	11.28
	simStructWalk	0.68	0.65	0.72	0.64	13.15	12.50	12.60
	RW + LW	0.68	0.64	0.74	0.65	18.65	11.35	11.67
	simStructWalk + LW	0.76	0.76	**0.81**	0.74	12.58	11.60	**10.50**

Bold entries represent the best accuracy for each model

Table 7 Classification and Regression results on the DBpedia-AU University dataset

Model	Method	Classification				Regression		
		NB	KNN	SVN	C4.5	KNN	LR	M5
CBOW 500	RW	0.48	0.90	0.89	0.88	54.25	12.70	12.56
	Rdf2Vec	0.72	0.89	0.29	0.92	45.67	12.44	12.30
	simStructWalk	0.69	0.74	0.75	0.72	38.57	18.65	6.28
	RW + LW	0.63	0.93	**0.95**	0.93	52.85	12.05	9.57
	simStructWalk + LW	0.78	0.82	0.93	0.86	38.36	17.06	**6.08**
SG 500	RW	0.55	0.83	0.87	0.87	55.59	12.54	11.37
	Rdf2vec	0.71	0.89	0.29	**0.92**	45.76	12.09	11.93
	simStructWalk	0.72	0.77	0.79	0.81	47.22	14.50	08.05
	RW + LW	0.67	0.91	0.93	0.89	53.68	11.55	**6.50**
	simStructWalk + LW	0.78	0.91	**0.92**	0.86	35.68	12.75	7.20

Bold entries represent the best accuracy for each model

5.2.2 Experimental results

As discussed in Section 4.1, for the publications dataset, the title, and abstract were used as informative data to summarize the paper. Thus, we assigned a new identity to the abstract of the paper, as in Section 4.1. We set the number of walks as 4 and depth as 7 on both datasets. Figures 8 and 9 show the accuracy of the KNN classification task for the DBLP and CiteSeerX datasets, respectively. To train sequences to the vectors, we used Skip-gram with five hundred dimensions and a window size of five. For the only RW, rdf2vec, and only simStructWalk cases, the sequences were only generated by neighborhoods but aggregated to the NL. Thus, any two papers having similar abstracts can be connected. For each case, the algorithm that combined with the NL had more accuracy than without the NL.

5.3 Experimental results summary

In summary, we conducted tests using eight datasets of various sizes and types. We created sets of sequences for each dataset using various techniques, and we trained the embedding using two different types of language modeling: Skip-gram and bag-of-word. We first discuss the accuracy of a group of RW, simStructWalk, and rdf2vec. Results from classification and regression tasks indicated that simStructWalk is generally much more accurate than RW. In contrast to rdf2vec, which in some cases provided higher accuracy, simStructWalk consistently produced fewer sequences than other methods. RW and rdf2vec must use 500 walks per node, and our simStructWalk only provides 100 walks per node. Our goal is to generate meaningful sequences. Therefore, if the accuracy of simtructWalk is not bad, we accept that result and employ these sequences to generate latent walks. Second, we combined those three methods to latent walk. The results demonstrate that using latent walk results in an average 30% improvement in accuracy over the non-latent walk method. The effectiveness of keeping textual literals as nodes in the classification task was then assessed. As a result, a set of sequences containing textual literals gives the mode a well-trained set. The classification's precision has greatly increased.

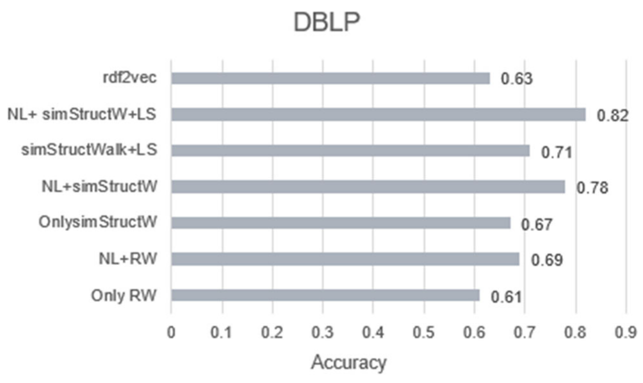


Fig. 8 Accuracy results of the experiment on the DBLP dataset

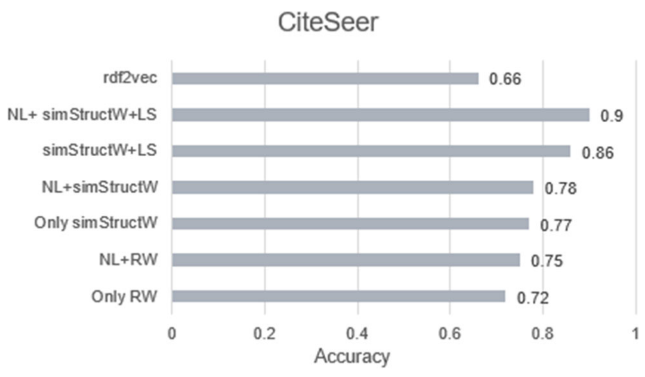


Fig. 9 Accuracy results of the experiment on the CiteSeerX dataset

6 Conclusion

In this study, we proposed three techniques to enhance the quality of graph walk-generated sequences. To generate the set of similar sequences for the similar nodes we first proposed (1) a new concept of similar entities in which trade-offs are made between similar outgoing edges and outgoing nodes, and (2) a new structural similarity that calculates the similarity between two entities in each sequence. By following these similarities, we provided the generated sequences such that similar entities have similar sequences. Second, the proposed method combines sequences with the same semantics to create latent sequences for the RDF node's semantic property that cannot be produced by traversing the graph. Finally, unlike some related works, we used the meaning of literal values to make them valuable nodes for training embedding rather than removing them. Our experimental evaluations demonstrate that our methods provide efficient sequences with good accuracy for both classification and clustering tasks.

In future work, we intend to propose more rules on the RDF graphs in subsequent work to diversify the types of latent sequences. To unitize all the valuable data in the RDF dataset, we retain not only textual literals but also other types such as number and time.

Acknowledgements This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. IITP-2022-2021-0-00859, Development of a distributed graph DBMS for intelligent processing of big graphs and No.RS-2022-00155911, Artificial Intelligence Convergence Innovation Human Resources Development (Kyung Hee University))

Declarations

Conflict of Interests The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Hogan A, Blomqvist E, Cochez M, d'Amato C, Melo GD, Gutierrez C, Kirrane S, Gayo JEL, Navigli R, Neumaier S et al (2021) Knowledge graphs. ACM Comput Surv (CSUR) 54(4):1–37
2. Wang M, Qiu L, Wang X (2021) A survey on knowledge graph embeddings for link prediction. Symmetry 13(3):485
3. Biswas R (2020) Embedding based link prediction for knowledge graph completion. In: Proceedings of the 29th ACM international conference on information & knowledge management, pp 3221–3224
4. Rossi A, Barbosa D, Firmani D, Matinata A, Merialdo P (2021) Knowledge graph embedding for link prediction: A comparative analysis. ACM Trans Knowl Discov Data (TKDD) 15(2):1–49
5. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:1301.3781
6. Pennington J, Socher R, Manning C (2014) GloVe: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1532–1543
7. Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 701–710
8. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 855–864
9. Ristoski P, Paulheim H (2016) RDF2Vec: RDF graph embeddings for data mining. In: International semantic web conference. Springer, pp 498–514
10. Cochez M, Ristoski P, Ponzetto SP, Paulheim H (2017) Biased graph walks for RDF graph embeddings. In: Proceedings of the 7th international conference on web intelligence, mining and semantics. ACM, p 21
11. Cochez M, Ristoski P, Ponzetto SP, Paulheim H (2017) Global RDF vector space embeddings. In: International semantic web conference. Springer, pp 190–207
12. Guan N, Song D, Liao L (2019) Knowledge graph embedding with concepts. Knowl-Based Syst 164:38–44
13. Xu M (2021) Understanding graph embedding methods and their applications. SIAM Rev 63(4):825–853
14. Ribeiro LF, Saverese PH, Figueiredo DR (2017) struc2vec: learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 385–394
15. Gesese GA, Biswas R, Alam M, Sack H (2021) A survey on knowledge graph embeddings with literals: Which model links better literal-ly? Semantic Web 12(4):617–647
16. Kristiadi A, Khan MA, Lukovnikov D, Lehmann J, Fischer A (2019) Incorporating literals into knowledge graph embeddings. In: International semantic web conference. Springer, pp 347–363
17. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119
18. Devlin J, Chang M-W, Lee K, Toutanova K (2018) BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
19. Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3:1137–1155
20. Krbec P (2006) Language modeling for speech recognition of Czech
21. Mikolov T, Kopecky J, Burget L, Glembek O et al (2009) Neural network based language models for highly inflective languages. In: 2009 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 4725–4728
22. Nechaev Y, Corcoglioniti F, Giuliano C (2017) Linking knowledge bases to social media profiles. In: Proceedings of the symposium on applied computing. SAC '17. Association for Computing Machinery, New York, pp 145–150
23. Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O (2013) Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems, pp 2787–2795
24. Wang Z, Zhang J, Feng J, Chen Z (2014) Knowledge graph embedding by translating on hyperplanes. In: AAAI, vol 14, pp 1112–1119
25. Lin Y, Liu Z, Sun M, Liu Y, Zhu X (2015) Learning entity and relation embeddings for knowledge graph completion. In: 29th AAAI conference on artificial intelligence

26. Cappuzzo R, Papotti P, Thirumuruganathan S (2019) Local embeddings for relational data integration. arXiv:1909.01120
27. Biswas R, Sack H, Alam M (2021) MADLINK: attentive multihop and entity descriptions for link prediction in knowledge graphs. Semantic Web (Preprint), 1–24
28. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Advances in neural information processing systems 27
29. Reimers N, Gurevych I (2019) Sentence-BERT: sentence embeddings using siamese BERT-networks. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP). Association for Computational Linguistics, pp 3982–3992
30. Cappuzzo R, Papotti P, Thirumuruganathan S (2020) Creating embeddings of heterogeneous relational datasets for data integration tasks. In: Proceedings of the 2020 ACM SIGMOD international conference on management of data, pp 1335–1349
31. Le Q, Mikolov T (2014) Distributed representations of sentences and documents. In: International conference on machine learning, pp 1188–1196. PMLR
32. Aguilar J, Salazar C, Velasco H, Monsalve-Pulido J, Montoya E (2020) Comparison and evaluation of different methods for the feature extraction from educational contents. Computation 8(2):30
33. Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: Graph processing in a distributed dataflow framework. In: 11th {USENIX} symposium on operating systems design and implementation ({OSDI} 14), pp 599–613
34. Lehmann J, Isele R, Jakob M, Jentzsch A, Kontokostas D, Mendes PN, Hellmann S, Morsey M, Van Kleef P, Auer S et al (2015) DBpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web 6(2):167–195
35. Ristoski P, De Vries GKD, Paulheim H (2016) A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: International semantic web conference. Springer, pp 186–194
36. Paulheim H (2012) Generating possible interpretations for statistics from linked open data. In: Extended semantic web conference. Springer, pp 560–574
37. Ristoski P, Paulheim H, Svátek V, Zeman V (2015) The linked data mining challenge 2015. In: KNOW@ LOD
38. Ristoski P, Paulheim H, Svátek V, Zeman V (2016) The linked data mining challenge 2016. In: (KNOW@ LOD/CoDeS)@ ESWC