Triple2Vec: Learning Triple Embeddings from Knowledge Graphs

Valeria FIonda[1] and Giuseppe Pirró[2]

[1] Department of Mathematics and Computer Science, University of Calabria, Italy
[2] Department of Computer Science, University of Rome La Sapienza, Italy

Abstract. Graph embedding techniques allow to learn high-quality feature vectors from graph structures and are useful in a variety of tasks, from node classification to clustering. Existing approaches have only focused on learning feature vectors for the nodes in a (knowledge) graph. To the best of our knowledge, none of them has tackled the problem of embedding of graph edges, that is, knowledge graph triples. The approaches that are closer to this task have focused on homogeneous graphs involving only one type of edge and obtain edge embeddings by applying some operation (e.g., average) on the embeddings of the endpoint nodes. The goal of this paper is to introduce Triple2Vec, a new technique to directly embed edges in (knowledge) graphs. Triple2Vec builds upon three main ingredients. *The first* is the notion of line graph. The line graph of a graph is another graph representing the adjacency between edges of the original graph. In particular, the nodes of the line graph are the edges of the original graph. We show that directly applying existing embedding techniques on the nodes of the line graph to learn edge embeddings is not enough in the context of knowledge graphs. Thus, we introduce the notion of triple line graph. *The second* is an edge weighting mechanism both for line graphs derived from knowledge graphs and homogeneous graphs. *The third* is a strategy based on graph walks on the weighted triple line graph that can preserve proximity between nodes. Embeddings are finally generated by adopting the SkipGram model, where sentences are replaced with graph walks. We evaluate our approach on different real world (knowledge) graphs and compared it with related work.

Keywords: Triple Embedding, Knowledge Graphs, Embeddings, Walks.

1 Introduction

In the last years, learning graph representations using low-dimensional vectors has received attention as viable support to various (machine) learning tasks, from node classification to clustering [3]. Approaches like DeepWalk [13], node2vec [6] and their variants strive to find node representations that preserve structural relations in the learned space. These approaches only focus on *homogeneous networks*, that is, networks (e.g., social networks) including only one type of edge. Another strand of research focused on embedding nodes in knowledge graphs (aka heterogeneous information networks) characterized by several distinct types of nodes and edges [18]. Notable approaches are rdf2vec [16], metapath2vec [4], and JUST [8]. One common denominator of both homogeneous and knowledge graph based embedding approaches is the usage of language model techniques. The idea is to consider sequences of nodes in a graph (i.e., random walks) as analogous to sentences in a document; then, the node sequences are fed into models like Skip-gram [10] to learn the final node embeddings. Despite the variety of available embedding techniques, to the best of our knowledge, *there is no technique that focuses on the embedding of triples (edges) of (knowledge) graphs.* The closest attempt we are aware of has been done by node2vec, where edge embeddings are learned by applying some operators (e.g., average) to the embeddings of the nodes at edge endpoints. This is insufficient in our context for several reasons. First, this approach has only considered homogeneous graphs; it is not clear how to behave when nodes are linked by more than an edge as in the case of knowledge graphs. Second, it is sub-optimal as it does not directly learn edge embeddings. Third, it does not embed labeled edges (i.e., triples).

The goal of this paper is to devise novel techniques to directly learn edge embeddings from both homogeneous and knowledge graphs. This sets three main challenges. *The first challenge is about how to go from node embeddings to edge embeddings.* One way to approach the problem could be to perform some (algebraic) manipulation on the endpoints of an edge; nevertheless, it is not clear how to behave in the context of knowledge graphs, where nodes may have multiple edges (i.e., predicates that reflect different relation perspectives) interlinking them. As an example, in Fig. 1 (a) Lauren Oliver and Americans are linked by two different edge types. To tackle this first challenge, we build upon the notion of *line graph* of a graph [19]. In its basic definition, the line graph has as nodes the edges of the original graph (nodes of the line graph are identified by the corresponding edge endpoints) while an edge is added between nodes if they share a common endpoint. This notion has been extended to both directed and multigraphs. As an example, the directed line graph obtained from the knowledge graph in Fig. 1 (a) is shown in Fig. 1 (b), where the two copies of the node Lauren Oliver, Americans correspond to the triples having nationality and citizenship as a predicate, respectively. It would be tempting to directly apply embedding techniques to the nodes of the directed line graph to obtain edge embeddings. However, we detect two main problems. The first is that it is impossible to discern between the two triples encoded by the nodes Lauren Oliver and Americans. The second is that the directed line graph is disconnected and as such, it becomes problematic to learn triple embeddings via random walks. Therefore, we introduce the notion of *triple line graph* $\mathcal{G}_L$ of a knowledge graph G; here, nodes are the triples of G and *an edge is introduced between nodes in* $\mathcal{G}_L$ *whenever the triples form G they represent share an endpoint.* This guarantees that $\mathcal{G}_L$ is connected if G is connected. The triple line graph for the graph in Fig. 1 (a) is shown in Fig. Fig. 1 (c).

However, the $\mathcal{G}_L$ alone is still not enough; in general, $\mathcal{G}_L$ has a much denser structure than G. *This introduces the second challenge related to the fact that high degree nodes in G get over-represented, in terms of the number of edges, in*

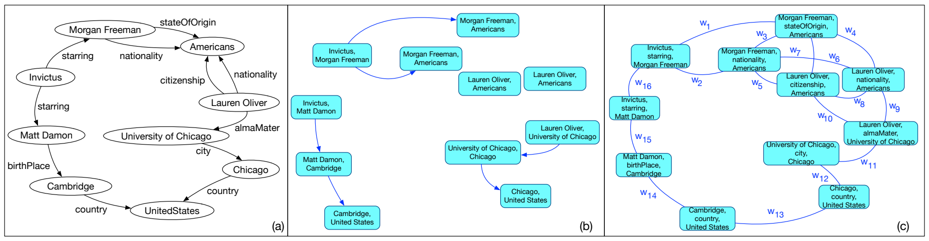

Morgan Freeman
stateOfOrigin
Americans
starring
nationality
Invictus
citizenship
nationality
Lauren Oliver
starring
almaMater
Matt Damon
University of Chicago
city
Chicago
birthPlace
Cambridge
country
country
UnitedStates
(a)
Invictus, Morgan Freeman
Morgan Freeman, Americans
Morgan Freeman, Americans
Lauren Oliver, Americans
Lauren Oliver, Americans
Invictus, Matt Damon
Lauren Oliver, University of Chicago
University of Chicago, Chicago
Matt Damon, Cambridge
Chicago, United States
Cambridge, United States
(b)
Morgan Freeman, stateOfOrigin, Americans
Invictus, starring, Morgan Freeman
Morgan Freeman, nationality, Americans
Lauren Oliver, citizenship, Americans
Lauren Oliver, nationality, Americans
Invictus, starring, Matt Damon
Lauren Oliver, almaMater, University of Chicago
University of Chicago, city, Chicago
Matt Damon, birthPlace, Cambridge
Chicago, country, United States
Cambridge, country, United States
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10} w_{11} w_{12} w_{13} w_{14} w_{15} w_{16}
(c)

Fig. 1: A knowledge graph (a), its directed line graph and (c) its triple line graph.

$\mathcal{G}_L$. To tackle this seconds challenge, we introduce two mechanisms to weight the edges of $\mathcal{G}_L$. The first, specific for *knowledge graphs,* assigns weights on the basis of predicate relatedness [14]. The weight of an edge between nodes of $\mathcal{G}_L$ is equal to the semantic relatedness between the predicates in the triples of G represented by the two nodes. As an example, in Fig. 1 (c) the weight of the nodes of $\mathcal{G}_L$ (M. Damon, birthPlace, Cambridge) and (Cambridge, country, United States) will be equal to the relatedness between birthPlace and country. The second mechanism, specific for *homogeneous graphs*, leverages the centrality of nodes in the original graph G. The weight of an edge between the nodes of $\mathcal{G}_L$ (u, i) and (i, j) is computed as a function of the flow centrality of u, i, and j in G.

The third challenge consists of how to compute the edge embeddings from the weighted $\mathcal{G}_L$. To this hand, we generate truncated random walks, in the form of sequences of nodes, on the weighted triple line graph. Note that weights based on semantic relatedness, for knowledge graphs, will bias the random walker to obtain similar contexts for nodes in the weighted triple line graph linked by related predicates. The underlying idea is that similar context will lead to similar embeddings. Finally, the walks are fed into the Skip-gram model [10], which will give the embeddings of the nodes of the weighted triple line graph that correspond to the embeddings of the original edges (for homogeneous graphs) or triples (for knowledge graphs). We assembled all the ingredients in Triple2Vec, which, to the best of our knowledge is the first approach that focuses on the problem of learning *edge embeddings from (knowledge) graphs*. We believe that edge embeddings can open up a novel class of downstream applications that go beyond that based on node embeddings. We are going to discuss applications in the area of edge classification and clustering.

Contributions and Outline. We tackle the problem of learning edge embeddings from (knowledge) graphs. To the best of our knowledge, this is the first work in this direction. We make the following main contributions:

– We introduce the notion of triple line graph that extends the notion of line graph to knowledge graphs.
– We introduce two different weighting mechanisms for triple line graphs. The first, suitable for knowledge graphs, assigns weights based on the relatedness between predicates in triples. The second one, suitable for homogeneous

graphs, weights the edges of the triple line graph based on the centrality of the nodes of the original graph.
- We introduce for the first time the notion of triple embedding, that is, a technique that can learn embeddings from triples in knowledge graphs.
- We describe novel application scenarios and compare our approach with related work.

The remainder of the paper is organized as follows. We introduce some preliminary definitions in Section 2. In Section 3, we introduce the notion of triple line graph of a knowledge graph along with an algorithm to compute it. Section 4 describes the Triple2Vec approach to learn triple embeddings from both homogeneous and knowledge graphs. In Section 5 we discuss an experimental evaluation. Related work is dealt with in Section 6. We draw some conclusions and sketch future work in Section 7

2 Preliminaries

A Knowledge Graph (G) is a kind of heterogeneous information network. It is a node and edge labeled directed multigraph $G{=}(V_G, E_G, T_G)$ where V is a set of uniquely identified vertices representing entities (e.g., D. Lynch), E_G a set of predicates (e.g., director) and T a set of triples of the form (s, p, o) representing directed labeled edges, where $s, o \in V$ and $p \in E_G$. Homogeneous graphs are represented as $G{=}(V_G, E_G)$, where E_G is a set of edges of the form (i, j). In what follows we will use the term edge to refer to both triples (for knowledge graphs) and edges for homogeneous graphs when it is clear from the context.

2.1 The Line Graph of a Graph

We introduce the notion of line graph of a graph starting with the case of undirected graphs. The idea of the line graph $\mathcal{G}_L$ of an undirected graph G is to represent adjacency information between the edges of G. More formally:

Definition 1 *Given an undirected graph $G = (V_G, E_G)$, where V_G (resp., E_G) is the sets of node (resp., edges), its line graph $\mathcal{G}_L = (V_L, E_L)$ is such that: (i) each node of $\mathcal{G}_L$ represents an edge of G; (ii) two vertices of $\mathcal{G}_L$ are adjacent if, and only if, their corresponding edges in G have a node in common.*

Starting from $G = (V_G, E_G)$ it is possible to compute the number of nodes and edges of $\mathcal{G}_L = (V_L, E_L)$ as follows: *(i)* the number of nodes of $\mathcal{G}_L$ is equals to the number of edges of G, i.e., $|V_L| = |E_G|$; *(ii)* $|E_L| \propto \frac{1}{2}\sum_{v \in V_G} d_v^2 - |E_G|$, where d_v denotes the degree of the node $v \in V_G$.

Example 2 *Each node of the line graph in Fig. 2 (d) is labeled with the endpoints of the corresponding edge in the original graph Fig. 2 (a). For instance, the node* a,b *in Fig. 2 (d) corresponds to the edge between the vertices* a *and* b *in Fig. 2 (a). The node* a,b *in Fig. 2 (d) is adjacent to the node* a,c *since the corresponding edges share the endpoint* a *and to the vertices* c,b*,* b,d *and* b,e *since the corresponding edges share the endpoint* b *(see Fig. 2 (a)).*

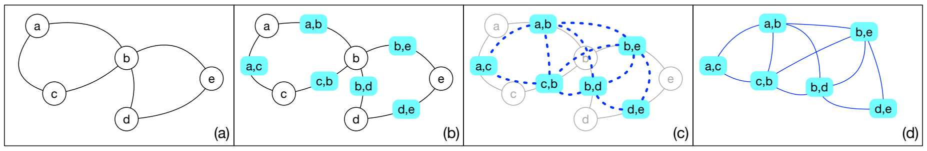


Fig. 2: A undirected graph (a), its line graph (d); construction steps (b)-(c).

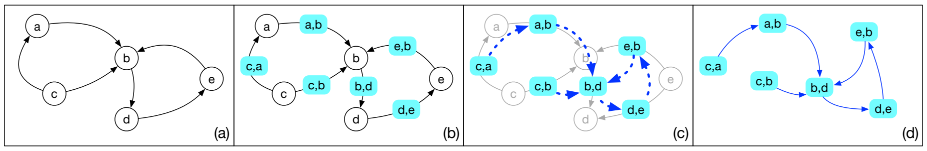


Fig. 3: A directed graph (a), its line graph (d); construction steps (b)-(c).

The concept of line graph has been extended to other types of graphs, including multigraphs and directed graphs. On one hand, the extension to multigraphs adds a different node in the line graph from each edge of the original multigraph. On the other end, if G is directed the corresponding line graph $\mathcal{G}_L$ will also be directed; in particular, its vertices are in one-to-one correspondence to the edges of G and its edges represent two-length directed paths in G.

Definition 3 *Given a directed graph $G = (V_G, E_G)$, its line graph $\mathcal{G}_L = (V_L, E_L)$ is a directed graph such that: (i) each node of $\mathcal{G}_L$ represents an edge of G; (ii) two vertices of $\mathcal{G}_L$, say* a,b *and* c,d*; an edge connects* a,b *and* c,d *iff,* b=c.

Example 4 *In Fig. 3 (d), each node of the directed line graph is labeled with the endpoints of the corresponding edge in the original directed graph. For instance, the node labeled* a,b *in Fig. 3 (d) corresponds to the edge from the node* a *to the node* b *in Fig. 3 (a). The node* a,b *in Fig. 3 (d) has an outgoing edge to the node* b,d *since the corresponding edges share the intermediate endpoint* b*, meaning that the edge from* a,b *to* b,d *encodes the directed two-length path from* a *to* d *in G (Fig. 3 (a)).*

3 The Triple Line Graph of a Knowledge Graph

The most natural way to apply the notion of line graph to knowledge graphs, that are directed labeled multigraphs, would be to apply Definition 3. However, this would lead to counter-intuitive behaviors. Consider the graph G in Fig. 4 (a). Fig. 4 (b) shows in blue nodes of the directed line graph $\mathcal{G}_L$, obtained by applying Definition 3, that are in one-to-one correspondence to the edges of G. Fig. 4 (c) shows how the directed edges are added to $\mathcal{G}_L$ to obtain the final directed line graph shown in Fig. 4 (d). At this point, three main issues arise. First, the standard definition associates to each node of the directed line graph the two

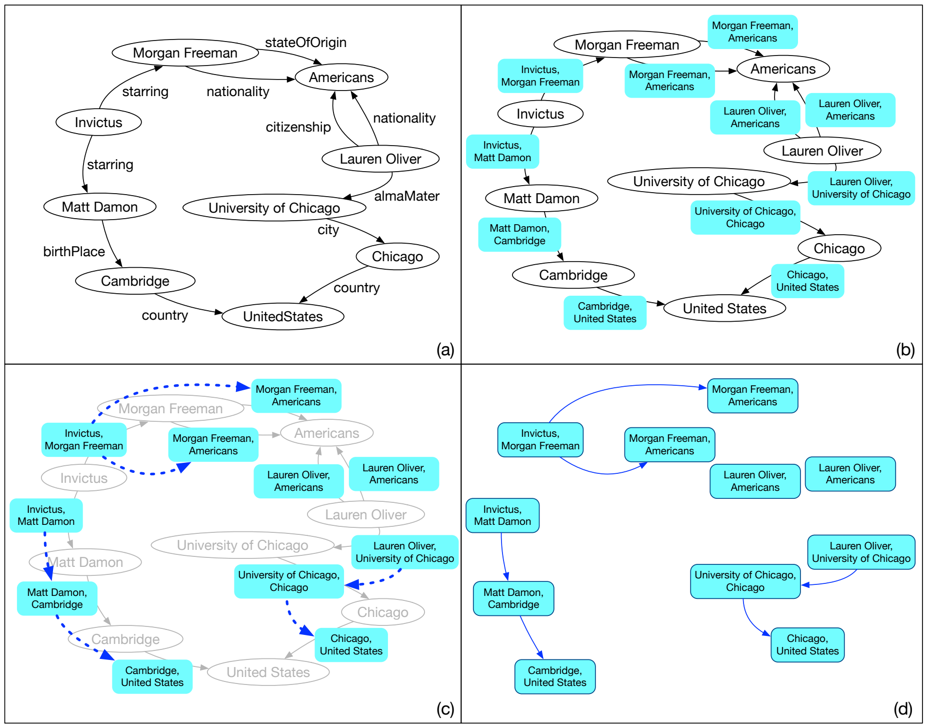


Fig. 4: A knowledge graph (a), its triple line graph (d); construction steps (b)-(c).

endpoints of the corresponding edge; however, the edge labels in a knowledge graph give a semantic to the corresponding edge, which is completely lost if only the endpoints are considered. Second, the edges to be added to the line graph are computed by considering their direction. This disregards the fact that edges in G witness some semantic relation between their endpoints (i.e., entities) that can be interpreted bidirectionally. As an example, according to Definition 3, the two nodes (Lauren Oliver, Americans) in $\mathcal{G}_L$ remain isolated since the corresponding edges do not belong to any two-length path in G (see Fig. 4 (c)-(d)). However, consider the triple (Lauren Oliver, nationality, Americans). While the traversal of the edge from Lauren Oliver to Americans serves the purpose of stating the relation nationality, the traversal of the edge in the opposite direction states the relation is nationality of. Hence, *in the case of knowledge graphs, two nodes of the line graph must be connected by an edge if they form a two-length path in the original knowledge graph no matter the edge direction, as the semantics of edges can be interpreted bidirectionally*. Finally, triples (via predicates) encode some semantic information, and the desideratum is to preserve this semantics when connecting two nodes (i.e., triples of G) in the line graph. Because of these issues, we introduce *triple line graphs*, a novel notion of line graph suitable for KGs.

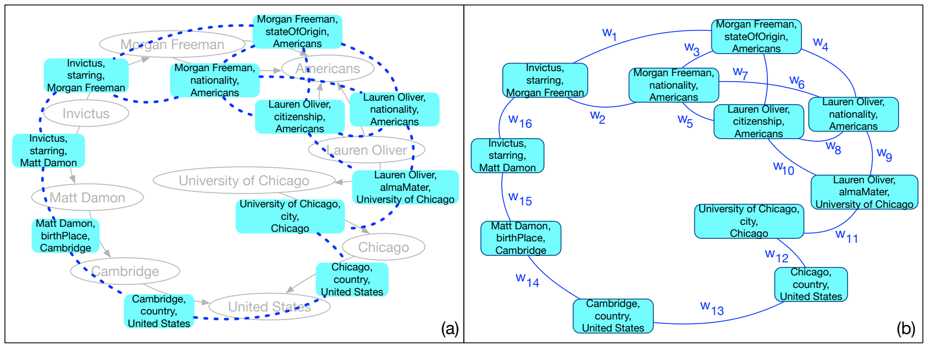

Morgan Freeman, stateOfOrigin, Americans
Morgan Freeman
Americans
Invictus, starring, Morgan Freeman
Morgan Freeman, nationality, Americans
Invictus
Lauren Oliver, citizenship, Americans
Lauren Oliver, nationality, Americans
Invictus, starring, Matt Damon
Lauren Oliver
University of Chicago
Lauren Oliver, almaMater, University of Chicago
Matt Damon
University of Chicago, city, Chicago
Matt Damon, birthPlace, Cambridge
Chicago
Chicago, country, United States
Cambridge
Cambridge, country, United States
United States
(a)
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12 w13 w14 w15 w16
(b)

Fig. 5: Subfigure (b) shows the triple line graph $\mathcal{G}_L$ associated to the knowledge graph G in Fig. 5 (a). Subfigure (a) shows the correspondence between the node of $\mathcal{G}_L$ and the triples of G.

Definition 5 *Given a knowledge graph $G = (V_G, E_G, T_G)$, the associated triple line graph $\mathcal{G}_L = (V_L, E_L, w)$ is such that: (i) each node of $\mathcal{G}_L$ represents an edge of G; (ii) two vertices of $\mathcal{G}_L$, say s_1, p_1, o_1 and s_2, p_2, o_2, are adjacent if, and only if, $\{s_1, o_1\} \cap \{s_2, o_2\} \neq \emptyset$; (iii) the function w associates a weight in the range $[0, 1]$ to each edge of $\mathcal{G}_L$.*

Given a Knowledge Graph $G = (V_G, E_G, T_G)$, its associated triple line graph $\mathcal{G}_L$ has $|T_G|$ nodes and each node of the original knowledge graph G, that is involved in k triples, creates k edges in the triple line graph.

Example 6 *Consider the graph G reported in Figure 4 (a). Fig. 5 (b) shows the associated triple line graph $\mathcal{G}_L$, while in Figure 5 (a) it is possible to identify the correspondence between the nodes of $\mathcal{G}_L$ and the triples in G. In particular, each node of the triple line graph is labeled with the subject predicate and object of the corresponding triple in G. For instance, the node labeled* Invictus, starring, Matt Damon *in Fig. 3 (b) corresponds to the triple* (Invictus, starring, Matt Damon) *in Fig. 4 (a). The nodes* Invictus, starring, Matt Damon *and* Invictus, starring, Morgan Freeman *in Fig. 5 (b) are connected by an edge since the subject of the triple corresponding to the former node is also subject of the triple of the latter. Moreover, such an edge has associated the weight w_{16} that should reflect the fact that the triples corresponding to the two endpoints share the same predicate* starring.

3.1 Computing Triple Line Graphs

We now describe an algorithm (outlined in Algorithm 1) to compute the triple line graph $\mathcal{G}_L$ of a knowledge graph G. This is at the core of Triple2Vec for the computation of triple embeddings. After initializing the set of nodes and edges of $\mathcal{G}_L$ to the empty set (line 1), the algorithm iterates over the triples of the input G and add a node to $\mathcal{G}_L$ for each visited triple (lines 2-3), thus inducing a bijection from the set of triples of G to the set of nodes of $\mathcal{G}_L$. Besides, if two triples share a node in G then an edge will be added between the corresponding nodes of $\mathcal{G}_L$ (lines 5-14). In particular, the data structure $I(s)$ (line 6) keeps track, for each node s of G, of the triples in which s appears as subject or object (lines 7-8). Since such triples correspond to nodes of the triple line graph, by iterating over pairs of triples in $I(s)$ it is possible to add the desired edge between the corresponding nodes of $\mathcal{G}_L$ (lines 9-10). By scrutinizing this algorithm, one can see that there can be a large number of edges in the triple line graph. Therefore, we introduce a generic edge weighting mechanism (line 11). We are going to describe edge weighting mechanisms for both knowledge graphs and homogeneous graphs in Section 4.1. We remark that in the case of homogeneous graphs, the computation of line graphs is done by applying the standard algorithm.

```
Input  : Knowledge Graph $G$
Output: $\mathcal{G}_L$: the Triple Line Graph associated to $G$
 1: $\mathcal{G}_L = \{\emptyset, \emptyset, \emptyset\}$
 2: for all $(s, p, o)$ in $G$ do
 3:     add the node $s, p, o$ to $\mathcal{G}_L$
 4: end for
 5: for all $s \in G$ do
 6:     $I(s) = \emptyset$
 7:     for all $(s, p, o)$ (resp., $(o, p, s)$) in $G$ do
 8:         add $s, p, o$ (resp., $o, p, s$) to $I(s)$
 9:         for all pair $n, n'$ in $I(s)$ do
10:             add the edge $(n, n')$ to $\mathcal{G}_L$
11:             set $w(n, n') =$ computeEdgeWeight$(n, n')$
12:         end for
13:     end for
14: end for
15: return $L(G)$
```

Algorithm 1: BuildTripleLineGraph (G)

By inspecting Algorithm 1, we observe that $\mathcal{G}_L$ can be computed in time $\mathcal{O}(|T|^2 \times costWeight)$, where $costWeight$ is the cost of computing the weight between nodes in $\mathcal{G}_L$ (i.e., triples in G).

4 Triple2Vec: Learning Triple Embeddings

We now describe Triple2Vec, which is the first approach for learning triple embeddings from knowledge graphs. Triple2Vec includes four main phases: (i) *building of the triple line graph* (outlined in Section 3); (ii) *weighting of the triple line graph edges* (outlined in Section 4.1); (iv) *computing walks* on the weighted triple line graph, described in Section 4.2, and (v) *computing embeddings* via the Skip-gram model, described in Section 4.3.

4.1 Triple Line Graph Edge Weighting

We have mentioned in Section 3.1 that the number of edges in the (triple) line graph can be large. This structure is much denser than that of the original graph and may significantly affect the performance of the walk generation strategy. To remedy this drawback, we introduce edge weighting functions (line 11 Algorithm 1) for both knowledge and homogeneous graphs.

Relatedness Based Edge Weights for Knowledge Graphs. The desideratum is to come up with a strategy to compute walks so that the neighborhood of a triple will include triples that are semantically related. We leverage a predicate relatedness measure [14], which looks at the co-occurrence of each pair of predicates p_i and p_j in the set of triples T and weights it by the predicate popularity [14]. Given two predicates p_i and p_j, $Rel(p_i, p_j)$, $R(p_i, p_j)$ is obtained by computing the cosine between their respective vectors. In terms of edge weights, the more related predicates in the triples representing two nodes in the triple line graph are, the higher the weight of the edge between these nodes. Driving the walks via relatedness allows to capture both the graph topology in terms triple-to-triple relations (i.e., edges in the triple line graph) and semantic proximity in terms of relatedness between predicates in triples.

Centrality-based Edge Weights for Homogeneous Graphs. For the case of homogeneous graphs (e.g., social networks), we introduce an edge weighting mechanism for $\mathcal{G}_L$ that relies on the notion of current-flow betweenness [2]. This measure characterizes the importance of each node in G in terms of the number of times it lies on a path between two other nodes (note that it extends the notion of betweenness centrality which focuses on shortest paths only). Therefore, a walker on G would be directly affected by the current-flow betweenness of each node. To reflect the same behavior on the line graph $\mathcal{G}_L$, the edge weighting mechanism assigns a weight to the edge from $n_p = i, j$ to $n_q = j, k$ in $\mathcal{G}_L$ (representing a path from i to k passing through j in G) proportional to the weighted mean among the current-flow betweenness centrality of the nodes i, j and k in the graph G. More formally, $w(n_p, n_q) = \alpha \cdot cb(i) + \beta \cdot cb(j) + \gamma \cdot cb(k)$ with $\alpha + \beta + \gamma = 1$ and $cb(x)$ being the current-flow centrality of the node x, with $x \in \{i, j, k\}$.

4.2 Computing Walks

Triple2Vec leverages a language model approach to learn the final edge embeddings. As such, it requires a "corpus" of both nodes and sequences of nodes similarly to word embeddings techniques that require words and sequences of words (i.e., sentences). To obtain the corpus from the graph, we leverage truncated graph walks. The idea is to start from each node of $\mathcal{G}_L$ (representing an edge of the original graph) and provide a context for each of such node in terms of a sequence of other nodes. Although walks have been used by previous approaches for both homogeneous (e.g., Deepwalk [13], node2vec [6]) and knowledge (e.g., metapath2vec, JUST [8]) graphs, none of them has tackled the problem of computing edge embeddings.

4.3 Computing Embeddings

Once the "corpus" (in terms of the set of walks $\mathcal{W}$) is available, the last step of the Triple2Vec workflow is to compute the embeddings of the nodes of $\mathcal{G}_L$ that will correspond to the embeddings of the edges of the input graph G. The embedding we seek can be seen as a function $f : \mathcal{V}_L \to R^d$, which projects nodes of the weighted triple line graph $\mathcal{G}_L$ into a low dimensional vector space, where $d \ll |\mathcal{V}_L|$, so that neighboring nodes are in close proximity in the vector space. For every node $u \in \mathcal{V}_L$, $N(u) \subset \mathcal{V}_L$ is the set of neighbors, which is determined by the walks computed as described in Section 4.2. The co-occurrence probability of two nodes v_i and v_{i+1} in a set of walks $\mathcal{W}$ is given by the softmax function using their vector embeddings e_{v_i} and $e_{v_{i+1}}$:

$$p((e_{v_i}, e_{v_{i+1}}) \in \mathcal{W}) = \sigma(e_{v_i}^T e_{v_{i+1}}) \tag{1}$$

where σ is the softmax function and $e_{v_i}^T e_{v_{i+1}}$ is the dot product of the vectors e_{v_i} and $e_{v_{i+1}}$ As the computation of (1) is computationally demanding [6], we use negative sampling to training the Skip-gram model. Negative sampling randomly selects nodes that do not appear together in a walk as negative examples, instead of considering all nodes in a graph. In particular, if a node v_i appears in walk of another node v_{i+1}, then the vector embedding e_{v_i} is closer to $e_{v_{i+1}}$ as compared to any other randomly chosen node. The probability that a node v_i and a randomly chosen node v_j *do not* appear in a random walk starting from v_i is given by:

$$p((e_j, e_i) \notin \mathcal{W}) = \sigma(-e_{v_i}^T e_{v_j}) \tag{2}$$

For any two nodes v_i and v_{i+1}, the negative sampling objective of the Skip-gram model to be maximized is given by the following objective function:

$$\mathcal{O}(\theta) = \log \sigma(e_{v_i}^T e_{v_{i+1}}) + \sum_{j=1}^{k} \mathbb{E}_{v_j}[\log \sigma(-e_{v_i}^T e_{v_j})], \tag{3}$$

where θ denotes the set of all parameters and k is the number of negative samples. For the optimization of the objective function, we use the parallel asynchronous stochastic gradient descent algorithm [15].

5 Experiments

In this section, we report on an experimental evaluation of Triple2Vec and comparison with related work. We describe the datasets and the experimental setting in Section 5.1. Then, we report experiments on knowledge graphs in Section 5.3 and on homogeneous graphs in Section 5.4.

5.1 Datasets and Experimental Setting

We discuss experiments on both knowledge graphs and homogeneous graphs. For knowledge graphs, we used the following three real-world data sets. In DBLP

Table 1: Datasets used.

Dataset	**|V|**	**|E|**
DBLP	16K	52K
Foursquare	30K	83K
Yago	22K	89K
Karate	34	78
Les Miserables	77	254
USA Power Grid	5K	6.6K

[7], there are 4 different kinds of edges linking authors to papers, papers to venues, papers to papers, and papers to topics. In addition, authors are labeled with one among four labels (i.e., database, data mining, machine learning, and information retrieval). Foursquare [20] includes four different kinds of entities, that is, users, places, points of interests and timestamps. Each point of interest has also associated one among 10 labels. Yago, described in [7], includes 5 types of edges interlinking movies to directors, actors and so forth. Moreover, each movie is assigned one or more among 5 available labels.

5.2 Systems and Parameter Setting

As Triple2Vec is the first approach to tackle the problem of embedding edges in (knowledge) graphs, there is an intrinsic difficulty in finding competitors. Therefore, we considered some existing approaches to learn edge embeddings. *For homogeneous graphs*, edge embeddings were computed by aggregating the embeddings of the endpoint nodes. As the average gave the best results, in what follows we only discuss this aggregation mechanism. *For knowledge graphs*, we adopted the same methodology. Nevertheless, in this case, this strategy leads to counter-intuitive behaviors since nodes can be connected by more than one edge in general (see Section 3). This further underlines the need for specific edge embedding approaches like Triple2Vec, which can handle these situations by directly embedding triples in KGs. Triple2Vec has been implemented in Python[3] using the networkx[4] and gensim[5] libraries to handle graph computations and learn embeddings, respectively. For the evaluation, we considered the following baselines implemented in the StellarGraph library[6].

[3] Link omitted to preserve the anonymity of the review
[4] `https://networkx.github.io`
[5] `https://radimrehurek.com/gensim`
[6] `https://www.stellargraph.io`

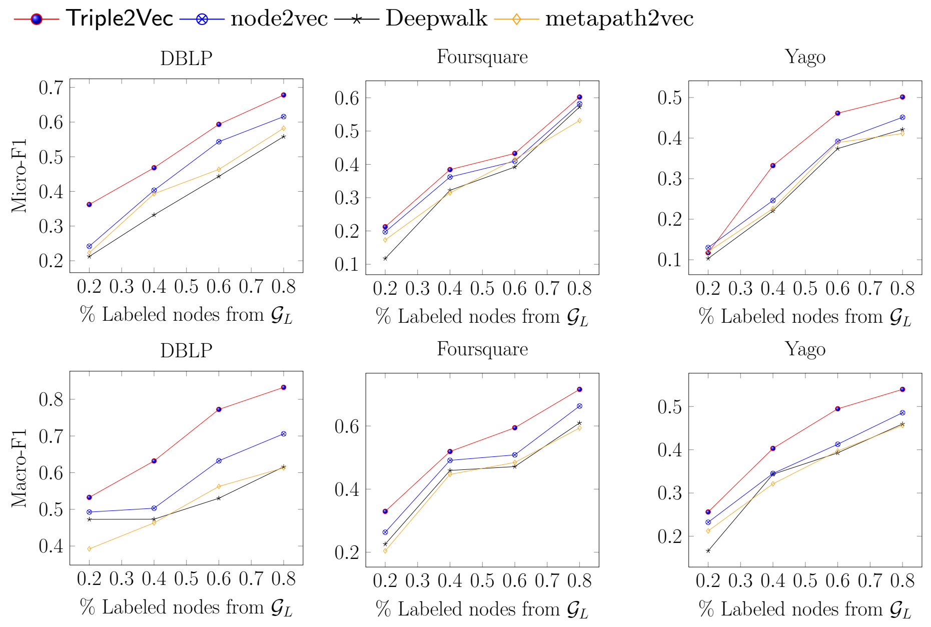
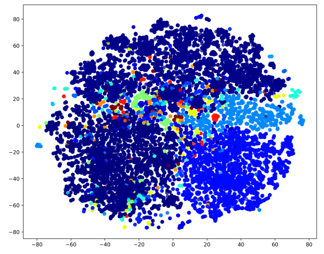


Fig. 6: Triple classification results in terms of micro and macro F1.

- **DeepWalk**: it learns node embeddings via random walks fed into the Skip-gram model. As this approach was originally designed for homogeneous graphs, we applied it on knowledge graphs by ignoring node and edge types.
- **node2vec**: it improves upon DeepWalk in both the way random walks are generated and in the objective function optimization (it uses negative sampling). We set the parameters specific to this algorithm (i.e., p and q) to the best values reported in [6]. For the same reason as Deepwalk, we apply node2vec by ignoring node and edge types.
- **Metapath2vec**: it has been defined to work on knowledge graphs. It takes as input one or more *metapaths* (i.e., sequences of node types) to generate walks fed into a variant of the Skip-gram model. For the evaluation, we used the metapaths used in previous evaluations [4,8].

For sake of space, in what follows, we only report the best results obtained by setting the parameters as follows: the number of walks per node n=10, maximum walk length $L = 100$, the window size (necessary for the notion of context in the Skip-gram model) $w = 10$. Moreover, we used d=128 as a dimension of the node embeddings for DBLP, Foursquare and Yago and d=32 for the other datasets. The number of negative samples Γ is set to 10 for all methods in all experiments. All results reported are the average of 10 runs.


(a) Triple2Vec

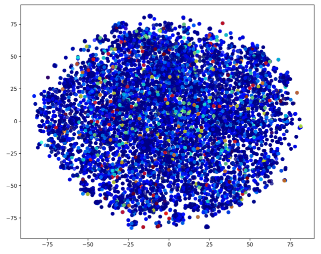

(b) metapath2vec

Fig. 7: Triple embedding visualization of the DBLP dataset.

5.3 Evaluation on Knowledge Graphs

To the best of our knowledge Triple2Vec is the first approach to specifically tackle the problem of embeddings triples from heterogeneous graphs like knowledge graphs. As such, there was no benchmark available. In order to construct a benchmark for the evaluation, we labeled the triples of the datasets by propagating the available labels. Specifically, for DBLP the 4 author node labels available have been propagated to paper nodes by following authorship links; then, from papers nodes to topic and venue nodes. For Yago, movie nodes were labeled with 12 labels that have been propagated to actors, musicians and directors by following actedIn, wroteMusicFor and directed edges, respectively. Finally, points of interest for FourSquare were labeled with 10 labels that have been propagated to places, users and timestamps by following locate, perform and happendAt edges, respectively. At the end of this label propagation step, each node of G is labeled with a subset of the initial labels. To propagate these labels to the nodes of $\mathcal{G}_L$, we considered the union of the sets of labels associated with the node endpoints of the corresponding triples in G represented by the nodes of $\mathcal{G}_L$. Finally, to each different subset of labels, we assigned a different label. We now report on the evaluation on two different tasks.

Triple Classification. In order to carry out this task, we trained a one-vs-rest Logistic regression model, giving as input the triple embeddings along with their labels (the labels of the node of $\mathcal{G}_L$). Then, we compute the Micro-F1 and Macro-F1 scores by varying the percentage of training data. Results are reported in Fig. 6. We observe that Triple2Vec consistently outperforms the baseline. This is especially true in the DBLP and Yago datasets. We also note that metapath2vec performs worse than node2vec and deepwalk, despite the fact that the former has been proposed to work on knowledge graphs. This may be explained by the fact that the metapaths used in the experiments, while being able to capture node embeddings, fail short in capturing edge (triple) embeddings.

Triple Clustering and visualization. To have a better account of how triple embeddings are placed in the embedding space, we used t-SNE [9] to obtain a

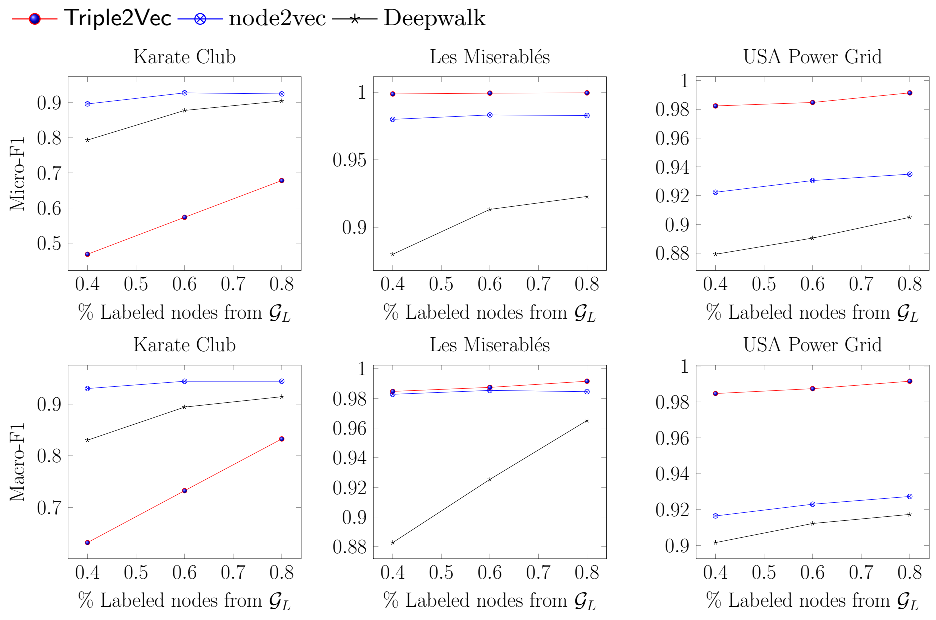


Fig. 8: Edge classification results in terms of Micro and Macro F1.

2-d representation of the triple embeddings (originally including d dimensions) obtained from Triple2Vec and metapath2vec, which is the only one among the competitors, designed for knowledge graphs. Results are shown in Fig. 7 on DBLP. We observed similar trends in the other datasets. We note that while Triple2Vec is able to clearly identify groups of triples (i.e., triples labeled with the same labels), metapath2vec offers a less clear perspective. We can explain this behavior with the fact that Triple2Vec defines a specific strategy for triple embeddings based on the notion of semantic proximity, while triple embeddings metapath2vec have been obtained from the endpoint nodes.

5.4 Evaluation on Homogeneous Graphs

For the case of homogeneous graphs, we compared Triple2Vec with Deepwalk and node2vec only. This is because metapath2vec requires metapaths as input that are not available in homogeneous graphs. The evaluation was carried as follows. For each graph, we first found node communities by using a modularity-based algorithm that does not return overlapping communities [12]. Then, for each community, returned as a set of nodes, we identify the set of intra-community edges and labeled each of such edges with the id of the community it belongs to. **Edge Classification.** As for the case of knowledge graphs, we trained a one-vs-rest Logistic regression model, giving as input the edge embeddings and the labels (the community they belong to). Results are reported in Fig. 8. We can notice that even in this case Triple2Vec performs better than the baselines. In

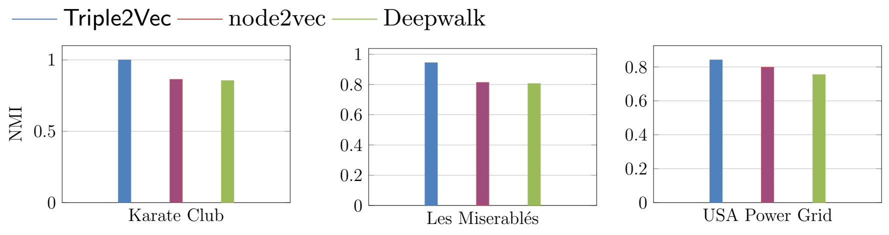


Fig. 9: Clustering results.

particular, the difference with the baselines becomes clearer when moving to larger networks (from left to right in Fig. 8).

Edge Clustering. We also evaluate the performance of the systems on a clustering task. In particular, we ran the K-means algorithm giving the edge embeddings as input. Then, we compute the Normalized Mutual Information (NMI). Results are reported in Fig. 9. Again, Triple2Vec performs better than the competitors. We want to mention the fact that we are reporting the best performance of node2vec and Deepwalk in terms of the aggregation mechanism on node embeddings. On the contrary, Triple2Vec does not requires any aggregation being focused on directly learning edge embeddings.

6 Related Work

There is a vast body of related research about graph embedding techniques for both homogeneous [3] and knowledge graphs [18]. In what follows we focus our attention on node embeddings that are the closest to our approach. Early work on node embeddings has focused on homogeneous graphs, that is, graphs having one type of node and edge only [6,13,17]. DeepWalk [13], node2vec [6] and LINE [17] are inspired by word2vec [10], which has been proposed to learn the vector representations of words that appear in a text corpus. These techniques sample a set of random walks (and, in particular, they differ in how they sample the walks) in the original graph that is fed to a Skip-gram model to generate the vector representation of nodes, so that two nodes that frequently co-occurr in a randomly sampled path will have similar embeddings. DeepWalk [13] generates short truncated random walks by uniformly sampling the starting node and the additional nodes from the neighbors of the last node visited. node2vec [6] generates biased random walks by using two parameters to control how fast the walk explores and leaves the neighborhood of a node. LINE [17] guides the generation of random walks by using 1-hop and 2-hop neighborhoods of nodes as such it learns two different latent representations of nodes.

Another strand of research has focused on heterogeneous graphs, where nodes and edges can have different types [4,5,8,16]. Here, the random walk generation for the Skip-gram model has been adapted to consider nodes and edge types.

RDF2Vec [16] focuses on computing node embeddings by using the continuous bag of words or a Skip-gram model. It computes two kinds of walks: subtrees up to a fixed depth k and breadth-first search walks(by uniformly sampling the nodes on the walks among the neighbors). metapath2vec [4] uses metapaths to guide the generation of walks, but it also proposes to use heterogeneous negative samples in the Skip-gram model for learning latent vectors of nodes. Hin2vec [5] is an evolution of metapath2vec, which considers multiple metapaths. JUST [8] provides a sampling strategy that balances both the presence of homogeneous, heterogeneous edges and the node distribution over different domains (i.e., node types) in the generated walks. Our approach is also different from TransE [1] and its variants, the goal of which is to learn knowledge graph embeddings to perform link prediction by providing both positive and negative input facts.

To the best of our knowledge, Triple2Vec is the first approach that focuses on embedding triples in knowledge graphs. We already mentioned that approaches line node2vec have proposed ways to learn embeddings for edges as some combination of embeddings of the edge endpoints (e.g., Hadamard product, average). Despite the fact that this approach is inherently sub-optimal, when applied to knowledge graphs it will lead to counter-intuitive behaviors. Indeed, all the triples involving the same pair of nodes will be given the same vector representation. Triple2Vec specifically focuses on learning triple embeddings guided by semantic proximity, which takes into account the semantics of edges. Finally, we also devised a novel edge weighting mechanism for homogeneous graphs.

7 Concluding Remarks and Future Work

We introduced the novel task of learning edges from (knowledge) graphs. While for homogeneous graphs, there have been some sub-optimal proposals [6], for knowledge graphs, the problem of learning triple embeddings, was never explored. We presented an elegant solution, which builds upon the notion line graph and extends it to knowledge graphs. We introduced semantic proximity as a way to place together triples expressed with related predicates close in the embedding space. We have also considered the case of homogeneous graphs, where node proximity is preserved via the notion of current-flow betweenness. The assembling of these novel ideas in the Triple2Vec system can pave the way to a novel class of applications based on triple embeddings. We have discussed in the experiments tasks related to triple classification and visualization, for knowledge graphs, and edge classification and community detection for homogeneous graphs. There is still a lot to be explored, from novel ways of imposing triple proximity (e.g., via constraints [11]) to novel applications like fact-checking.

References

1. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

2. U. Brandes and D. Fleischer. Centrality measures based on current flow. In *STACS*, pages 533–544.
3. H. Cai, V. W Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE*, 30(9):1616–1637, 2018.
4. X. Dong, N. V Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *CIKM*, pages 135–144, 2017.
5. T.-Y. Fu, W.-C. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*, pages 1797–1806, 2017.
6. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.
7. Z. Huang and N. Mamoulis. Heterogeneous information network embedding for meta path based proximity. *arXiv preprint arXiv:1701.05291*, 2017.
8. R. Hussein, D. Yang, and P. Cudré-Mauroux. Are meta-paths necessary?: Revisiting heterogeneous graph embeddings. In *CIKM*, pages 437–446. ACM, 2018.
9. L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.
10. T. Mikolov, I. Sutskever, K. Chen, G. S Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.
11. P. Minervini, L. Costabello, E. Muñoz, V. Nováček, and P.-Y. Vandenbussche. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *PKDD*, pages 668–683, 2017.
12. M. EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.
13. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.
14. G. Pirrò. Building relatedness explanations from knowledge graphs. *Semantic Web*, (Preprint):1–28.
15. B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.
16. P. Ristoski and H. Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *ISWC*, pages 498–514, 2016.
17. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.
18. Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.
19. D. Brent West et al. *Introduction to graph theory*, volume 2. Prentice Hall, 1996.
20. D. Yang, D. Zhang, and B. Qu. Participatory cultural mapping based on collective behavior data in location-based social networks. *ACM TIST*, 7(3):30, 2016.