



**POLITECNICO**  
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE



CentraleSupélec



# A Hybrid Approach for Embedding Knowledge Graphs

TESI DI LAUREA MAGISTRALE IN  
COMPUTER SCIENCE AND ENGINEERING  
PDS: ARTIFICIAL INTELLIGENCE

Author: **Elisa Mariani**

Student ID: 10632876

Advisor: Prof. Marco Brambilla

Co-advisors at LISN<sup>1</sup>: Prof. Nacéra Seghouani, Prof. Yue Ma

Academic Year: 2022-2023

---

<sup>1</sup>This thesis has been conducted at the LISN laboratory of Université Paris-Saclay, as part of the Alliance4Tech exchange program between Politecnico di Milano and Centrale-Supélec.



# Abstract

In today's digital era, knowledge graphs (KGs) provide a solution to manage the abundance of data by organizing information in a structured manner. They represent information by organizing it into "triples", which are relationships between distinct entities or concepts within the domain under consideration.

The most common issues plaguing KGs are incompleteness and the presence of noise, often resulting from the integration of data from various sources. Numerous procedures aiming to refine such KGs have been developed in recent years using embeddings, which are low-dimensional numerical representations of the entities and relationships within the KGs. Beyond refining KGs, these embeddings play a pivotal role as they make the data within KGs accessible to algorithms exclusively designed for numerical data processing, including some used in recommendation systems, natural language processing, and text analysis.

A significant challenge in the embedding creation process is the use of "negative triples", which represent relationships not contained in the KG. These are essential in KG embedding models, as they enable them to distinguish between relationships that exist within the KG and those that do not.

Upon analyzing solutions proposed in the literature and identifying areas for improvement, TransHI was conceived. This approach introduces an iterative method for generating high-quality embeddings. What sets TransHI apart from cutting-edge algorithms is its adoption of a hybrid approach in generating negative triples, amalgamating information from the KG's structure and its related ontologies, which are formal representations that delineate possible relationships within the KG's domain. This approach became necessary as ontologies might not encompass information pertaining to all entities and relationships within the graph. A hybrid approach that combines both ontologies and structure allows for a more uniform utilization of the KG's information. While the iterative training of embeddings has been previously explored in the literature, TransHI offers a novel mode of implementing it and a preprocessing phase that effectively facilitates the pipeline, especially in the case of KGs characterized by significant noise.

The preprocessing phase represents the initial step of the TransHI methodology. Leveraging the ontologies associated with KGs, this phase aims to minimize noise within KGs by eliminating potentially erroneous triples and augmenting the number of truthful triples. Subsequently, the method transitions into an iterative phase. In the first iteration, an algorithm is employed to train the embeddings, generating negative triples in a hybrid manner: utilizing both the KG's structure and its related ontology. After this initial training, a classification algorithm is used to pinpoint information that was not adequately learned during training and, therefore, is not well-represented by the generated embeddings. This information is then harnessed in the creation of effective negatives during subsequent training iterations.

The use of iterative training steps, combined with the hybrid approach employed during the initial training iteration, has proven to be particularly effective. Additionally, the preprocessing phase has shown to be highly beneficial, especially in refining noisy KGs.

The results highlighted better performances of TransHI over two state-of-the-art algorithms, TransE and TransOWL, in the link prediction task. However, a potential risk of overfitting emerged when there is excessive integration of ontological knowledge into the KG during the preprocessing phase. This risk can be mitigated by using a different algorithm exclusively for the initial training iteration, as the overfitting is observed only with the method used during that phase. For subsequent iterations, the embedding algorithm of TransHI remains suitable.

In summary, TransHI represents a promising advancement in the field of KG embeddings, offering a robust solution to the challenges characterizing contemporary embedding algorithms.

**Keywords:** Knowledge Graph, Ontology, Graph Embedding, Link Prediction, Supervised Training

## Abstract in lingua italiana

Nell'odierna epoca digitale, i grafi di conoscenza (KGs) offrono una soluzione per gestire l'abbondanza di dati, organizzando le informazioni in modo strutturato. Essi rappresentano le informazioni organizzandole in "triple", cioè in relazioni che intercorrono tra entità, individui distinti o concetti del dominio in esame.

Un'importante sfida nel processo di creazione degli embeddings è l'utilizzo di 'triple negative', ovvero triple che rappresentano relazioni non appartenenti al KG. Esse sono essenziali nei modelli di embedding di KGs, in quanto permettono loro di rappresentare la distinzione tra relazioni contenute nel KG e non.

Analizzando le soluzioni avanzate dalla letteratura e identificando aree di miglioramento, è stato concepito TransHI. Questo approccio introduce un metodo iterativo per generare embeddings di alta qualità. La peculiarità che contraddistingue TransHI rispetto agli algoritmi all'avanguardia è l'adozione di un approccio ibrido nella generazione delle triple negative, amalgamando informazioni provenienti dalla struttura del KG e dalle ontologie ad esso correlate, rappresentazioni formali che circoscrivono le relazioni possibili nel dominio del KG. Tale approccio si è reso necessario in quanto le ontologie possono non contenere informazioni relative a tutte le entità e relazioni all'interno del grafo e un approccio ibrido che combini ontologie e struttura consente un utilizzo più uniforme delle informazioni del KG. Seppure il training iterativo di embeddings sia già stato utilizzato in letteratura, TransHI propone una nuova modalità per implementarlo ed una fase di pre-processing che consente di utilizzare efficacemente la pipeline in caso di KGs caratterizzati da molto rumore.

Tale fase di preprocessing rappresenta la tappa iniziale della metodologia TransHI. Avvalendosi delle ontologie associate ai KGs, questa fase mira a minimizzare il rumore nei KGs, estirpando triple potenzialmente errate ed aumentando il numero di triple veritiere. Di seguito, il metodo procede entrando in una fase iterativa. Nella prima iterazione si adopera un algoritmo per addestrare gli embeddings che crea le triple negative in modo ibrido: sfruttando sia la struttura del KG che l'ontologia relativa. Dopo questo training iniziale, si sfrutta un algoritmo di classificazione per individuare le informazioni che

non sono state apprese adeguatamente durante il training e pertanto non sono rappresentate bene dagli embeddings generati. Queste informazioni vengono poi impiegate nella creazione di negative efficaci durante iterazioni successive di training.

L'utilizzo di steps di training in modo iterativo, combinato con l'approccio ibrido utilizzato durante la prima iterazione del training si è dimostrato particolarmente efficace. Anche la fase di pre-elaborazione si è manifestata assai proficua, in particolar modo nella raffinazione di KGs con molto rumore.

I risultati hanno evidenziato prestazioni migliori di TransHI rispetto a due algoritmi all'avanguardia, TransE e TransOWL, nel compito di previsione dei collegamenti. Tuttavia, è emerso un potenziale rischio di overfitting in presenza di un'eccessiva integrazione di conoscenza ontologica nel KG durante la fase di pre-elaborazione. Tale rischio può essere mitigato adottando un algoritmo differente esclusivamente per la prima iterazione dell'addestramento, dal momento che l'overfitting si manifesta soltanto con il metodo impiegato in questo step. Per le iterazioni successive, l'algoritmo di embedding di TransHI risulta efficace.

In sintesi, TransHI rappresenta un avanzamento promettente nel campo degli embeddings di KGs, proponendo una soluzione solida alle sfide che caratterizzano gli algoritmi di embedding contemporanei.

**Parole chiave:** Grafo di Conoscenza, Ontologia, Embedding di Grafi, Previsione dei Collegamenti, Addestramento Supervisionato

# Contents

<b>Abstract</b>	<b>i</b>
<b>Abstract in lingua italiana</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Preliminaries</b>	<b>5</b>
2.1 Graphs fundamentals . . . . .	5
2.1.1 Basic concepts . . . . .	5
2.1.2 Characteristics and properties . . . . .	6
2.1.3 Random walks for graph exploration . . . . .	8
2.1.4 Common graphs representations . . . . .	9
2.2 Knowledge graphs . . . . .	10
2.2.1 Definition . . . . .	10
2.2.2 Additional structures in KGs . . . . .	12
2.2.3 TBox and Abox . . . . .	14
2.2.4 Real-world examples . . . . .	15
2.3 Ontologies . . . . .	16
2.3.1 Definition and basics . . . . .	16
2.3.2 Axioms . . . . .	19
2.3.3 Ontology based reasoning - Entailment . . . . .	21
2.3.4 Ontology based reasoning - Consistency . . . . .	22
2.4 KG Refinement . . . . .	23
2.4.1 Missing edges in knowledge graphs . . . . .	23
2.4.2 Completion - Link prediction . . . . .	25
2.4.3 Correction - Fact validation and inconsistency repair . . . . .	25
2.5 Knowledge graph embeddings . . . . .	26

2.5.1	Embeddings basics . . . . .	26
2.5.2	Knowledge graph embeddings basics . . . . .	28
2.5.3	Embedding models basics . . . . .	28
<b>3</b>	<b>Related Work</b>	<b>31</b>
3.1	Two translational models: TransE and TransOWL . . . . .	31
3.2	Graph structure for negative triples creation . . . . .	36
3.3	Iterative approach for negative sampling . . . . .	38
<b>4</b>	<b>The new approach</b>	<b>41</b>
4.1	TransHI - The main contributions . . . . .	41
4.2	TransHI - A general overview . . . . .	43
4.3	TransHI - Preprocessing . . . . .	45
4.3.1	Inconsistencies Correction . . . . .	45
4.3.2	Ontology Axioms Entailment . . . . .	54
4.3.3	Positive Triples Augmentation . . . . .	57
4.4	Trans-HI - Training in first iteration . . . . .	58
4.4.1	Ontology-based negative triples . . . . .	63
4.4.2	Structure-based negative triples . . . . .	67
4.5	TransHI - Training Data Update . . . . .	71
4.6	TransHI - Training in next iterations . . . . .	74
<b>5</b>	<b>Experiments &amp; Evaluation</b>	<b>81</b>
5.1	Implementation and experimental settings . . . . .	81
5.2	Link prediction and evaluation metrics . . . . .	83
5.3	Overview of the three datasets . . . . .	85
5.4	Preprocessing results . . . . .	86
5.5	Training - first iteration results . . . . .	88
5.6	Training Data Update & next iterations results . . . . .	93
<b>6</b>	<b>Conclusions and future developments</b>	<b>97</b>
6.1	Contribution . . . . .	97
6.2	Future work . . . . .	98
	<b>Bibliography</b>	<b>101</b>



<b>A Appendix A</b>	<b>105</b>
<b>A Appendix B</b>	<b>107</b>
<b>List of Figures</b>	<b>111</b>
<b>List of Tables</b>	<b>113</b>



# 1 | Introduction

In the contemporary digital landscape, the sheer volume of data being generated is escalating at an unprecedented rate. The necessity to structure this surge of data to derive significant insights has never been more urgent. Within this context, knowledge graphs (KGs) have emerged as a potent solution, facilitating structured organization and querying of data. Within these graphs, information is systematically arranged into "facts" or "triples". Each fact delineates a specific relationship between two entities within a knowledge domain. For instance, the fact  $(Rome, Capital, Italy)$  elucidates a distinct relationship between the entities *Rome* and *Italy*. Notable exemplars of KGs include DBPedia, YAGO, FreeBase, NELL, and WikiData. Complementing the structure of KGs are ontologies. These articulate background knowledge about the domain that the KG encompasses, proffering a formal structure and delineating permissible relationships between entities.

Knowledge graphs (KGs) often arise from the integration of multiple data sources, a process typically facilitated by automated tools or crowd-sourcing. This mode of creation inherently predisposes such data structures to two primary challenges: incompleteness and noise. To address these issues, various KG refinement techniques have been developed. Among these are Link Prediction, a process aimed at identifying information that is likely accurate but absent from the KG, and Triple Classification, a procedure that ascertains the veracity or inaccuracy of information in relation to the domain knowledge encapsulated within the KG.

In recent years, these two refinement techniques have been developed leveraging a numerical vector representation of the information contained within KGs, known as embeddings. This numerical representation not only encapsulates the semantic information of the data in KGs but, due to its compact nature, proves especially advantageous for extensive KGs. The application of embeddings to represent KG information is not confined to refinement tasks alone. Translating the information within knowledge graphs into a numerical format has facilitated the integration of such data into a plethora of Machine Learning algorithms. Noteworthy among these are recommendation algorithms, which tailor sug-

gestions for e-commerce site customers and streaming platform users; Natural Language Processing (NLP) algorithms, enhancing natural language interpretation; and text analysis algorithms, including those for information retrieval, sentiment analysis, and automatic translation.

For the generation of embeddings, algorithms are employed with the primary objective of converting the knowledge encapsulated within KGs into a numerical format, while minimizing the loss of information inherent to their structure. The majority of these algorithms utilize training data comprising triples from the KGs, referred to as "positive triples" and "negative triples", the latter representing information not contained in the KG. Through this approach, embedding models learn to distinguish between factual and non-factual assertions, with the embeddings reflecting this distinction. The generation of negative triples serves as a distinguishing factor for each embedding algorithm. A foundational algorithm, TransE, proposed by Bordes et al. [7], generates negative triples randomly. A more advanced model, TransOWL, introduced by D'Amato et al. [10], leverages the background knowledge contained within ontologies to produce more effective negative triples compared to random ones. A distinct approach is adopted by Ahrabian et al. [1], where the intrinsic structure of the Knowledge Graph itself is harnessed.

The principal objective of this research is to enhance the prevailing state-of-the-art algorithms and generate high-quality embeddings that accurately capture the domain-specific information inherent in KGs. This is achieved by the design of a novel algorithm and the evaluation of its performance on benchmark datasets.

## Main Contributions

This work introduces not merely a new training algorithm but an entire iterative pipeline : TransHI. The distinctive feature of TransHI, compared to cutting-edge algorithms, is the adoption of a hybrid approach in generating negative triples, merging information from the structure of the KG and its related ontologies. Such an approach became necessary because ontologies might not contain information about all the entities and relationships within the KG. A hybrid approach that combines both ontologies and structure allows for a more uniform use of the KG's information. Although iterative training of embeddings has been previously explored in literature, TransHI introduces a novel method for its implementation and a preprocessing phase that enables effective use of the pipeline in the presence of highly noisy KGs.

This preprocessing phase constitutes the initial step of the TransHI pipeline. Leveraging the ontologies associated with the KGs, this phase aims to minimize noise in the KGs by eliminating potentially erroneous triples and augmenting the number of truthful triples.

Subsequently, the iterative segment of the pipeline encompasses a novel training algorithm for generating embeddings, followed by a "Training Data Update" phase that aims to discern which pieces of information have not been sufficiently captured by the produced embeddings. Notably, the algorithm employed for generating embeddings, uses a different approach in the first iteration with respect to the following ones. In the first iteration, the negative triples are created based on both ontologies and the structure of the KG. As for the following iterations, the identification of facts that were inadequately learned during the training phase, carried out during the "Training Data Update" phase, allows their use in a subsequent training iteration. By incorporating these facts in a new training phase, the focus is on the correction of embeddings involved in such triples.

The efficacy and potential constraints of this approach were empirically assessed on three real-world KGs using a link prediction algorithm. This evaluation was conducted to ascertain the proficiency of the embeddings, generated through the novel approach, in predicting the existence of facts that were not learned during the training phase.

TransHI's implementation is available as an open-source tool, promoting transparency and collaboration while fostering both academic research and practical applications.

## Thesis Outline

The structure of the thesis is as follows:

- Chapter 2 introduces all the theoretical prerequisites essential for understanding this work. Concepts such as graphs, knowledge graphs, ontologies, embeddings, and the foundational principles of embedding models are elucidated.
- Chapter 3 delves into the state-of-the-art methodologies employed for knowledge graph embeddings, which served as the foundation for the novel approach presented.
- Chapter 4 unveils the new approach, TransHI, initially providing an overview of the entire pipeline, followed by a detailed explanation of each step. This section offers insights into both the methodology employed and the theoretical rationale underpinning the decisions made.
- Chapter 5 showcases the evaluation phase of the embeddings producible through the newly introduced approach. Specifically, it begins by explaining the link prediction algorithm used to assess the quality of the generated embeddings. Subsequently, the empirical results stemming from the application of the new method on three real-world KGs are presented and scrutinized, highlighting the strengths and limitations of the approach.

- Chapter 6 offers concluding remarks on the efficacy of the method and introduces potential future endeavors that could build upon the presented work.

## 2 | Preliminaries

In the present section, an overview of the theoretical foundations related to graph theory, knowledge graphs, ontologies, embeddings and their applications is provided. These theoretical underpinnings form the basis of TransHI, the new framework that will be presented in the following sections.

### 2.1. Graphs fundamentals

Following is a list of fundamental concepts in graph theory.

#### 2.1.1. Basic concepts

**Graph-** A graph  $G$  is a mathematical structure used to represent relations between entities, called nodes or vertices. It is defined as the ordered couple  $(V, E)$  with  $V$  (or  $V(G)$ ) being the finite, nonempty set of nodes and  $E$  (or  $E(G)$ )  $\subseteq V \times V$  the set of edges (or arcs), representing relationships among them.

**Size of a graph-** The size of a graph is defined as the number of its edges,  $|E|$ .

**Labelled graph-** A "labelled graph" is characterized by the presence of one label assigned to each edge and/or vertex of the graph. A label is a piece of information (a string, a number or a more complex structure) assigned to a node or an arc to identify it and enrich it with context and meaning. When the labels are assigned only to edges, the graph is defined as "edge-labelled". Instead, when they are assigned only to vertices, it is "vertex-labeled".

**Simple and Multigraph-** A graph is "simple" when  $\forall v_i, v_j \in V, \exists$  at most one edge  $e = (v_i, v_j) \in E$ . Otherwise, it is called "multigraph".

**Oriented graph-** Graphs can be also classified based on the type of the edges: if they are ordered pairs of elements in  $V$ , the graph is said to be "oriented", "non-oriented" otherwise. In oriented graphs, each arc  $e = (v_1, v_2) \in E(G)$  is "outgoing" from  $v_1$  and "incoming" in  $v_2$ . Given an oriented graph  $G = (V, E)$ , the "non-oriented version" of

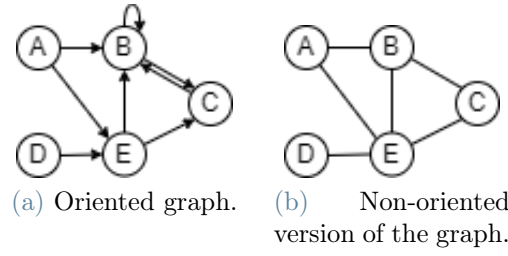


Figure 2.1: Representation of a graph and its non-oriented version.

$G$  is the graph obtained by ignoring the direction of the arcs and the self-loops (arcs connecting a vertex to itself) in it. In Fig. 2.1 there is a representation of an oriented graph (2.1a) and its undirected version (2.1b).

In a non-oriented graph  $G = (V, E)$ , given an entity  $e = (v_1, v_2) \in E(G)$  and two vertices  $v_1, v_2 \in V(G)$ , the edge has as "endpoints"  $v_1$  and  $v_2$  and it is said to be "incident" in them. Moreover, the two vertices are said to be "adjacent" and this relationship is symmetric. In an oriented graph instead, the edge  $e$  is "incident" from  $v_1$  in  $v_2$  and this relationship is not necessarily symmetric.

### 2.1.2. Characteristics and properties

Henceforth, the characteristics and properties of oriented graphs will be elucidated, as they are pertinent to the presentation of the new approach presented.

**Walk-** A "walk" in a non-oriented graph  $G = (V, E)$  is defined as a sequence of vertices  $(v_1, v_2, \dots, v_n)$  and edges  $(e_1, e_2, \dots, e_{n-1})$  such that  $e_i$  is incident in  $v_i$  and  $v_{i+1}$  for each  $i$  in  $1 \leq i \leq n-1$ . The "length" of a walk is defined as the cardinality of the set of edges that constitutes it. Considering two vertices  $v_1, v_2 \in V$ , it is said that  $v_2$  is "reachable" from  $v_1$  when a walk  $w$  between the two exists. In the case where all vertices of a walk are distinct, it is called a "trail" or "simple walk". If, instead, both the vertices and the arcs are all distinct, it is defined as "path". In oriented graphs, the direction of the arcs is also considered and not only their incidence in the vertices: in a walk with vertices  $(v_1, v_2, \dots, v_n)$  and arcs  $(e_1, e_2, \dots, e_{n-1})$ , each arc  $e_i$  will be exiting from  $v_i$  and entering into  $v_{i+1}$ . Furthermore, if the initial vertex  $v_1$  corresponds to the final vertex  $v_n$  and the length of the walk is at least 3, the walk is defined as "cycle".

**Distance between vertices-** In a graph  $G=(V,E)$ , the length of the shortest path between two vertices is defined as the "distance" between them. If a path between two points does not exist, their distance is considered infinite.

**Neighborhood-** A concept that is strictly related to the one of distance between nodes



is the concept of neighborhood. All the nodes that are adjacent to a vertex  $v \in V$  and that therefore have distance equal 1 from it are defined as its "neighbors". In the case of oriented graphs, this concept can be specialised into "In-neighbourhood" and "Out-neighbourhood". The former includes all vertices such that there is a direct arc outgoing from the neighbours to vertex  $v$ , and in the latter case the arc is outgoing from vertex  $v$ , entering the neighbours. On the other hand, "k-hops neighbors" of a vertex  $v$  are defined as the set of vertices that have a distance from it equal to  $k$ . It is important to emphasise the difference between this latter concept and that of "k-th neighbourhood". The "k-th neighbourhood" of a vertex includes all the entities that can be reached by a walk of length  $k$  from it: this allows the revisiting of vertices and arcs along the path. Thus, the main difference between the two is that the  $k$ -th neighbourhood of a vertex  $v$ , as opposed to the set of  $k$ -hop neighbours, may include vertices that are connected to  $v$  by a non-shortest path of  $k$  edges. A neighbourhood of a vertex  $v$  is termed "open" when it also contains the vertex itself, otherwise it is termed "closed".

**Degree of vertices-** In an undirected graph  $G$ , the "degree" of a vertex  $v \in V(G)$  is defined as the number of arcs incident on it. In the case of an oriented graph, this definition specialises in "incoming degree" and "outgoing degree": the number of arcs entering and leaving the vertex, respectively. Representing the number of connections of the individual vertices, the degree provides information on the centrality of the vertices and, more generally, on the structure of the graph.

**Density of graphs-** The "density" of a graph is defined as the ratio between the number of arcs and the number of vertices in it. In particular, in oriented graphs, it can be calculated as:  $D = \frac{|E|}{|V|(|V|-1)}$ . The maximum number of edges is reached when the graph is complete, i.e. each vertex is connected to all the others and it is equal to  $|V|(|V|-1)$ . The minimum number of arcs is 0, when the vertices are completely isolated. Therefore, the density is a value between 0 and 1. When the density of a graph approaches 1 (i.e.,  $|E| \approx |V|^2$ ), it is called "dense", when it approaches 0 (i.e.,  $|E| \ll |V|^2$ ), "sparse".

**Connectivity of graphs-** Given a graph  $G$ , two vertices  $v_1$  and  $v_2 \in V$  are said to be "connected" if there is a walk between them. The graph  $G = (V, E)$  is said to be "connected" if  $\forall v_1, v_2 \in V$  they are connected, otherwise "disconnected". In the case of oriented graphs, the graph is said to be "strongly connected" if the connection of each pair of vertices follows the direction of the arcs. Otherwise, it is said to be "weakly connected". Given a graph  $G = (V, E)$ , the "connectivity" of the graph is defined as the minimum number of vertices  $S$  such that  $G \setminus S$  is a disconnected graph.

**Clustering coefficient (CC) of vertices-** The connectivity of a graph provides a mea-

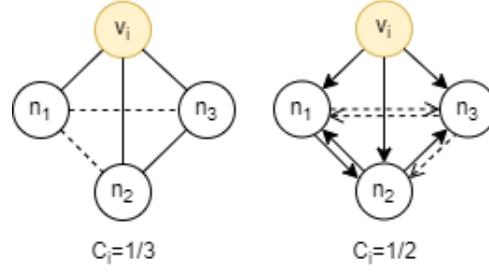


Figure 2.2: CC of node  $v_i$  in a non-oriented graph (left) and an oriented graph (right).

sure of the interconnection of nodes at a global level. A measure of the interconnection of nodes adjacent to a reference node is the local "clustering coefficient" (CC). Given the nodes close to a reference node, this index measures how interconnected they are by measuring the density at a local level. Given an oriented graph  $G = (V, E)$  and a vertex  $v_i$  with neighbors  $N_i$ , the clustering coefficient  $C_i$  can be calculated as follows:

$$C_i = \frac{2 \cdot |\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

In the case of an oriented graph, having  $|N_i|(|N_i| - 1)$  possible connections between neighbors, the clustering coefficient is calculated as:

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{|N_i|(|N_i| - 1)}$$

A high CC for a node suggests that its neighbours are highly interconnected, forming a densely connected area. In fact, this coefficient can be useful for identifying "clusters": areas of a graph characterised by a higher density (i.e. with a high number of edges relative to the number of vertices in these areas) than other areas of the graph.

Figure 2.2 shows a simple example of the computation of the CC relative to a node,  $v_i$ . In case of non-oriented graph,  $CC = (1/3)$  as the possible connections between the neighbors of vertex  $v_i$  are 3 but the one actually present is only 1. In the case of the oriented graph,  $CC = (1/2)$  as the connections actually present are 3 and the possible connections between the 3 neighbors are 6 (two for each pair of neighbors, with opposite directions).

### 2.1.3. Random walks for graph exploration

One of the techniques for exploring the structure of a graph, which is based on the concept of "neighbors", is called "**random walk**". Random walks are walks in which,

starting from an arbitrary vertex, at each step one moves from the selected vertex to a neighbour of the first, chosen with probabilistic techniques. If arcs have no cost (or all have equal cost), at each step, the next vertex is chosen uniformly at random. There are several implementation choices that may influence the behaviour of a random walk: one may choose to abort the walk after a finite number of steps, avoid specific nodes, use a probability for choosing a neighbour based on its structural characteristics such as its connectivity or degree. One can also choose to avoid returning to the immediately preceding node: this prevents the walk from being 'stuck' in frequent loops and is therefore used extensively in the context of exploring dense networks.

#### 2.1.4. Common graphs representations

The two most commonly used data structures for representing graphs are adjacency matrices and adjacency lists.

##### Adjacency matrices

In the case of a simple oriented graph  $G = (V, E)$ , the adjacency matrix  $M$  is a square binary matrix of order  $|V|$  such that,  $\forall v_i, v_j \in |V|$ :

$$M_{i,j} = \begin{cases} 1 & \text{if } e = (v_i, v_j) \in E(G) \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

The spatial complexity of adjacency matrices is  $O(|V|^2)$ , independently of the number of arcs. However, these structures allow one to quickly check whether an arc between two vertices  $v_i$  and  $v_j$  exists: one directly accesses the value in the relative matrix  $M_{i,j}$  and checks whether it is equal to 1 or not. Direct access corresponds to a complexity of  $O(1)$ . In contrast, to list all nodes adjacent to a node  $v_1$  the complexity becomes  $O(|V|)$  as the entire row  $M_{1,:}$  must be inspected. These structures may be inefficient in the representation of sparse graphs as they also represent (with value 0) non-existing arcs, even if the number of those actually present is  $|E| < |V|^2$ .

##### Adjacency lists

Another type of data structure for representing graphs that may be more suitable in the case of sparse graphs is adjacency lists. The adjacency list of a vertex  $v$  contains all the vertices adjacent to it. Thus, to represent a graph, one must use a list/array of adjacency lists, one for each vertex. A simple example of a graph adjacency list and adjacency

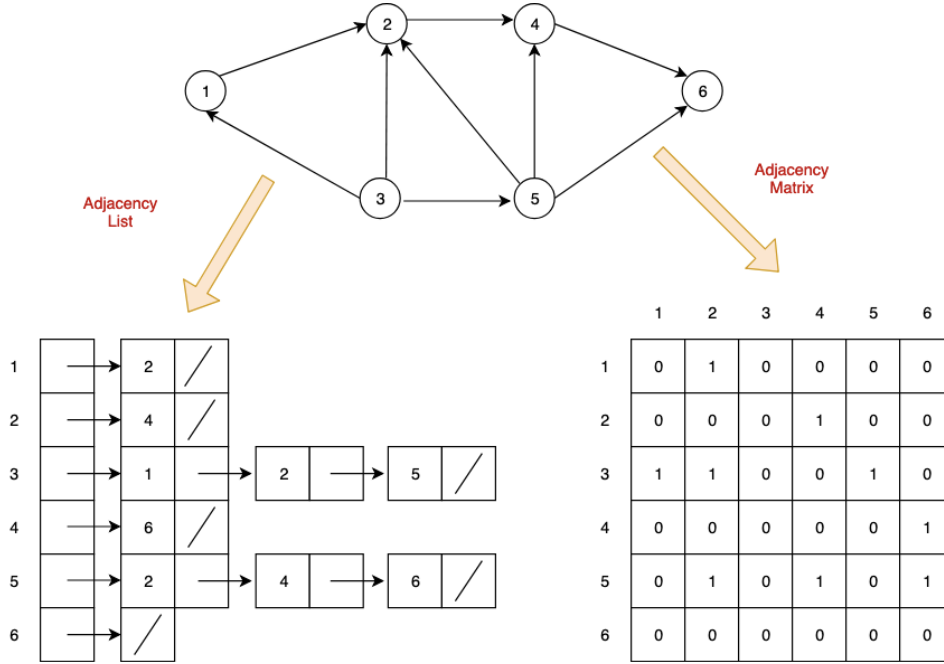


Figure 2.3: Example of adjacency matrix and adjacency list of an oriented graph.

matrix can be found in Figure 2.3<sup>1</sup>.

In these data structures, the spatial complexity depends on the amount of arcs present in the graph: it is equal to  $O(|V| + |E|)$  in the case of oriented arcs. The cost of checking whether an edge between two vertices exists or not is equal to  $O(|V|)$  and that of listing the neighbours of a given node only is linear to the degree of the vertex examined.

## 2.2. Knowledge graphs

### 2.2.1. Definition

A knowledge graph (KG) is an **edge-labeled oriented graph** used to represent information in a particular domain of knowledge. In a knowledge graph, the nodes correspond to real-world entities and the edges represent the semantic relationships between them. Formally, given a set of entities  $E$  and a set of relationships between them  $R$ , a knowledge graph  $K$  is defined as a subset of the cross-product  $E \times R \times E$ . In other words, it is possible to see  $K$  as a set of "**triples**", also known as "**facts**":

$$K = \{(h, r, t) \mid h, t \in E, r \in R\}$$

Each triple  $(h, r, t)$  is represented in the graph as two nodes,  $h$  and  $t$ , connected by a

<sup>1</sup>Figure from "<https://algorithmtutor.com/Data-Structures/Graph/Graph-Representation-Adjacency-List-and-Matrix/>"

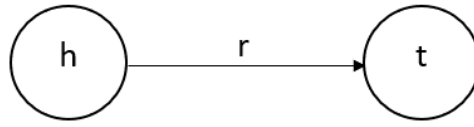


Figure 2.4: Generic triple  $(h,r,t)$  in a knowledge graph

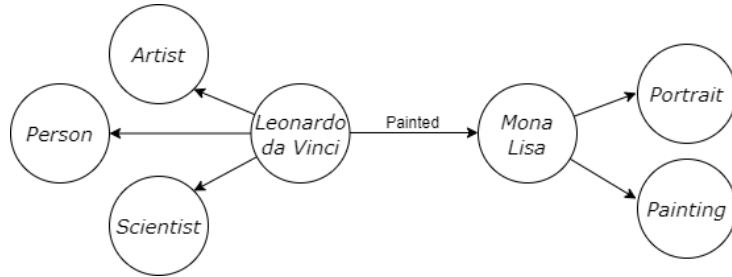


Figure 2.5: Example of a simple knowledge graph.

directed edge from  $h$  to  $t$  with label  $r$  (represented in Fig. 2.4). The nodes  $h$  and  $t$  can be defined as the **"head"** (or 'subject') and **"tail"** (or 'object') of the triple, respectively.

In the simple example of a knowledge graph depicted in Figure 2.5, it is possible to observe how the nodes "Leonardo da Vinci" and "Mona Lisa" are not merely connected by a link indicating a generic relationship between them: it bears a precise meaning that is easily interpretable by a human. Indeed, the link in the example clearly conveys that Leonardo da Vinci painted the Mona Lisa. There are no constraints regarding the type of "facts" that can be described with these structures. For instance, another link between the two entities of the example could be "IsAuthorOf" or "Created".

In today's digital era, the volume of data generated is increasing exponentially every day. Organizing this vast amount of data to extract valuable insights has become imperative. KGs play a pivotal role in this context. They offer a rich structure for representing a broad spectrum of data and their logical connections in a manner that is intuitively comprehensible to the human mind.

KGs possess a structured design, making the automation of data analysis and querying more straightforward than with unstructured databases, such as text-based ones.

In contrast to traditional databases characterized by a rigid structure, KGs exhibit enhanced flexibility. Modifying relationships or adding columns within a tabular structure can be resource-intensive. However, in KGs, the addition or removal of entities and relationships can be accomplished with ease. This distinction becomes especially significant when dealing with rapidly evolving data. Representing relationships between data in different tables can be inefficient (since it often requires costly JOIN operations) and might

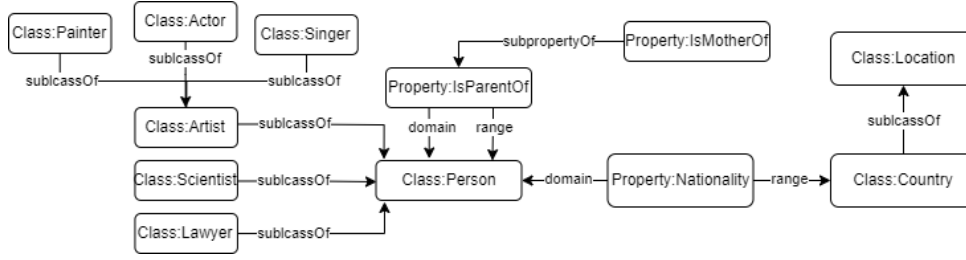


Figure 2.6: Semantic schema graph example.

result in creating predominantly empty columns or rows. KGs, by design, are tailored to model relationships between diverse entities, making them more suitable in scenarios where relationships among data are more important than the data within individual entities.

### 2.2.2. Additional structures in KGs

In knowledge graphs, the oriented edge-labeled data graph can be enhanced by the presence of additional structures to store knowledge. Among these structures, the discussion focuses on those that are crucial to understand the new approach: semantic schemas, identities and ontologies.

The first two will be discussed in the following sections, while ontologies will be addressed in Chapter 2.3.

#### The semantic schema

The semantic schema holds paramount importance. The semantic schema of a KG is intended to define a high-level structure that the graph should adhere to. It thus represents semantic information expressed as rules and conventions on data interconnection. A simple example is illustrated in Figure 2.6.

As long as the nodes of the KG are concerned, the schema provides semantic information about the "high-level terms" within the KG. These terms, also known as the "vocabulary" or the "terminology" of the KG, represent general and abstract concepts. They can be used to define "**classes**", categories that group nodes of the same type in the KG.

For example, in the simple KG depicted in Figure 2.7, the nodes "Leonardo da Vinci", "Albert Einstein", and "Marie Curie" can be grouped under the "Person" class as well as the "Scientist" class. One can also note that "scientist" is a category at a level below "person", so in the schema, it will be defined as a subclass. Similarly, in the example shown in Figure 2.6, the classes "Painter", "Actor", and "Singer" are subclasses of "Artist". The

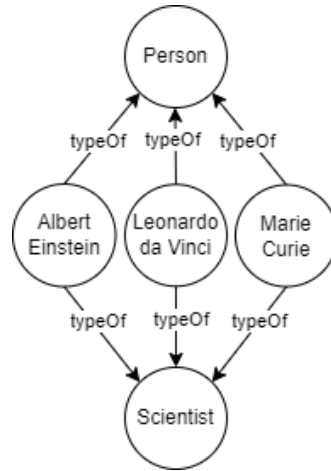


Figure 2.7: A simple KG with 2 classes and 3 instances.

latter, along with "Farmer" and "Lawyer", are subclasses of "Person". Finally, "Country" is a subclass of the class "Location".

Thus, the schema not only identifies the main categories of entities by defining the classes but also establishes a hierarchical structure among them.

As far as the relations of the KG are concerned, the schema defines them as "**properties**", and in this case, it also establishes a hierarchy among them. Beyond hierarchies, the schema defines both the domain and the range of these properties and in particular which entities can constitute the head and the tail of a particular relation, indicating the "permitted" classes.

In the example in Figure 2.6, the relationship "IsMotherOf" is a subproperty of "IsParentOf", and the "nationality" property has the class "Country" as its range and "Person" as its domain.

The semantic schema of a KG can be represented as a graph itself, with nodes equivalent to the classes and edges of the KG. The relations between those nodes constitute their relationships, such as the definition of the hierarchies between them.

The schema thus offers a lucid understanding of the data structure. It can be incorporated within the knowledge graph itself, established as an independent structure, or integrated into an ontology (presented in Chapter 2.3).

## Identities

The fundamental idea behind the concept of identity is that two entities might share the same name, necessitating a unique identifier to avoid ambiguities within the graph. This is particularly relevant when the graph emerges from the integration of multiple databases.

As exemplified by Hogan et al.[15], a node labeled "Santiago" might indicate different cities such as Santiago de Compostela or Santiago de Cuba. Therefore, it is imperative to have identifiers for each entity that are unique and independent of the KGs.

Within the context of the Semantic Web<sup>2</sup>, **URIs** (Uniform Resource Identifiers) and **IRIs** (Internationalized Resource Identifiers) are employed to uniquely identify entities and relationships within KGs. URIs and IRIs are text strings designed to uniquely identify a resource on the internet. The sole distinction between them is that IRIs can contain internationalized characters, whereas URIs do not. URLs (Uniform Resource Locators) serve a similar purpose, but they are unsuitable for identifying entities since they themselves could represent entities, introducing potential ambiguities.

IRIs and URIs are created by combining a string pertinent to specific entities with a prefix known as the "namespace". Namespaces delineate the entity's context. For instance, as depicted by the authors of [15], the namespace corresponding to the context of Chile might be "http://turism.cl/entity/", rendering the entity "http://turism.cl/entity/Santiago" unambiguously identifiable.

When the same entity is used in multiple contexts, it could be associated with several URIs/IRIs. This challenge can be addressed in two ways: by associating each entity with distinctive information, enabling the recognition of equivalent entities with different namespaces, or by employing the so-called "identity links", which are relationships "sameAs" between the two IRIs.

### 2.2.3. TBox and Abox

In knowledge graphs, the terms Tbox (Terminological Box) and Abox (Assertional Box) define two distinct types of information.

Comparable to the semantic schema discussed in the preceding section, the **Tbox** delineates a high-level structure that data should adhere to. The definitions of classes, properties, their hierarchies, and domain/range constraints evident in the semantic schema are examples of Tbox information. Beyond these, the Tbox may also encompass more intricate details regarding a knowledge graph's structure, such as those articulated by axioms in ontologies, as explored in Chapter 2.3.

Conversely, the **Abox** encompasses data relating to specific entities that belong to the real world within the knowledge graph. These are commonly referred to as "**instances**".

---

<sup>2</sup>"The Semantic Web is a proposed extension to the WorldWide Web (WWW) that aims to provide a common framework for sharing and reusing data across applications." [14]



For instance, consider the simple knowledge graph depicted in Figure 2.7. "Leonardo da Vinci", "Albert Einstein", and "Marie Curie" are instances of the classes "Person" and "Scientist". Facts containing these entities are examples of Abox information, even if they assign an instance to the respective class. If a knowledge graph is not integrated with its corresponding schema, it is entirely composed of facts within the Abox.

To denote the membership of an entity to a class in the triples of a KG, the relation "type" is employed. For clarity, in this study, all triples that use the "type" relation will be referred to as "typeOf" triples. Those with any other relation will be termed "noTypeOf" triples.

Together, the Tbox and Abox represent the domain of knowledge. While the Tbox outlines the classes and relationships that can exist within a knowledge graph and their interrelations, the Abox illustrates the lower-level structure, populating the knowledge graph with real-world entities interconnected by relationships that align with the semantics defined in the Tbox. These concepts allow for a distinction between structural information and instance-level data.

#### 2.2.4. Real-world examples

KGs are articulated using various standardized languages, which enable the description of concepts, properties, and relationships in a precise and unambiguous manner. Some of the most common include:

- **RDF (Resource Description Framework):** A standard from the World Wide Web Consortium (W3C) where information is represented in the form of triples.
- **OWL (Web Ontology Language):** Originally designed for ontology descriptions (discussed in Chapter 2.3). It can represent both conceptual structures and specific facts between individuals, making it suitable for describing KGs.
- **TURTLE (Terse RDF Triple Language):** A format for serializing RDF data. Unlike RDF/XML, which uses XML syntax for describing triples, TURTLE employs a clearer and more readable notation, making it one of the most preferred languages for KG descriptions.

While numerous small-scale KGs focus on specific themes, some encompass a vast array of topics and are not confined to a particular domain. Prominent open-source knowledge graphs include:

- **DBpedia:** A crowd-sourced, community-based KG aimed at extracting and struc-

turing data from Wikipedia pages.

- YAGO (Yet Another Great Ontology): A vast KG comprising data from DBPedia, Wordnet, and Geonames. Facts from Wikipedia pages are structured using WordNet hierarchies.
- Freebase: Originated not from information extraction but user collaboration. Acquired by Google in 2012, it remains a significant structured information source for the Google Knowledge Graph as of 2015.
- Wikidata: An initiative launched by the Wikimedia Foundation, enabling collaborative data curation, aiming to have a unified and standardized database as the foundation for other Wikimedia projects.
- OpenCyc: The open-source version of the Cyc database, initiated in 1984 with the goal of structuring human common-sense knowledge for automated use.
- NELL: A Carnegie Mellon University project employing continuous machine learning techniques to extract information from the internet. The acquired information is stored in NELL's KG, which grows continually as new information is learned.

For a more detailed comparison of these KGs, readers are referred to Färber et al. [33].

## 2.3. Ontologies

Ontologies and knowledge graphs are closely related concepts and are often used together in Semantic Web applications. Ontologies can serve as a foundation to build knowledge graphs or to integrate them with additional information, enhancing the precision of query and interpretation of results. Furthermore, knowledge graphs can also be used as a tool to populate and enrich ontologies with additional information from various sources, contributing to their completeness.

### 2.3.1. Definition and basics

Ontologies are formal and shared definitions of a vocabulary of terms and their interrelationships in a specific domain of interest. They serve to specify a formal conceptualization: to give a unique meaning to the terms that define the knowledge in a particular domain [12].

From a formal point of view, an ontology can be represented as a quadruple  $O = (C, R, A, I)$ , where:  $C$  is the set of concepts or classes,  $I$  is a set of instances,  $R$  is

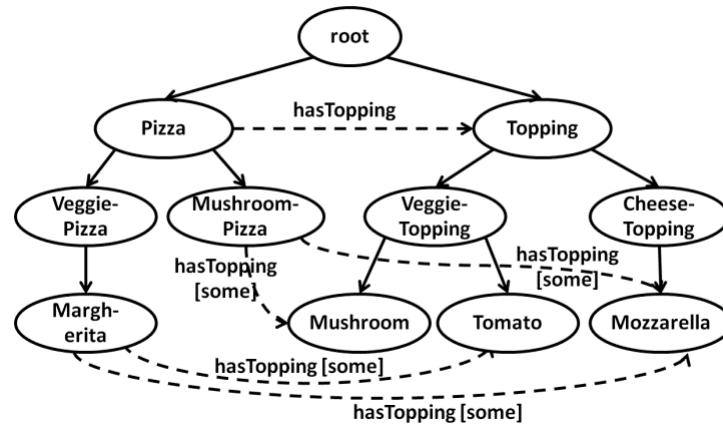


Figure 2.8: Example of a simple ontology.

the set of relations (or properties) and  $A$  is a set of axioms. The first three elements have been introduced in paragraphs 2.2.2 and 2.2.3. The axioms of an ontology define the formal properties of the concepts and relationships (e.g., the axioms may define whether a relationship is symmetric, transitive, reflexive, etc.). Furthermore, axioms can define constraints on concepts, for example, defining whether a certain concept can have a single instance or whether it can have many instances.

In Chapter 2.2.2, the discussion revolved around how a semantic schema is characterized by definitions of classes, properties, and the relationships among them. As previously introduced, this semantic schema can be embedded within an ontology and does not necessarily have to be a separate structure. As a matter of fact, ontologies serve as instruments that provide greater depth and detail. One distinction is that they encompass instances of a particular domain, not merely the classes and properties. They contain both Tbox and Abox, concepts discussed in Section 2.2.3. Another distinction lies in the expression of the structure and interconnection of classes. While a schema might operate within predefined relationships, an ontology offers a broader range of properties. As will be explored in Section 2.3.2, axioms can articulate more complex relationships between classes than those mapped out for semantic schemas. This capability allows for the crafting of a more detailed structure, representing the knowledge domain with increased precision.

An example of a simple ontology that is commonly used for demonstrative purposes represents knowledge about pizzas, their ingredients, and their properties. This ontology can be expressed in the  $\langle C, R, I, A \rangle$  format as follows:

- Concepts will be classes such as 'Pizza', 'Topping', 'Veggie-Pizza', 'Veggie Topping', 'Cheese-topping'.

- Relations will be, for example, "hasTopping".
- Instances of the Pizza concept will be, for example, "Margherita", "Mushroom-Pizza", and instances of the Topping concept will be "Tomato", "Mozzarella".
- Axioms will be, for example, "A pizza must have at least one topping".

A graphical representation of this ontology is presented in Figure 2.8 <sup>3</sup>.

Ontologies are created and manipulated using formal languages that allow exploiting the information contained in the ontology automatically in other applications. RDF (Resource Description Framework) and OWL (Web Ontology Language). Among the most prominent, three languages have previously been cited in the context of KGs: RDF, OWL, and Turtle. Notably, OWL offers an extensive array of logical constructs, including negation, union, intersection, and restriction, enabling the development of intricate ontologies.

With reference to the previously presented ontology example, a representative snippet of OWL code for its construction is now presented:

```
<!-- Define the classes -->
<Class rdf:about="#Pizza"/>
<Class rdf:about="#Topping"/>
<Class rdf:about="#VeggiePizza">
  <subClassOf rdf:resource="#Pizza"/>
</Class>
...

<!-- Define the instances -->
<NamedIndividual rdf:about="#Margherita">
  <rdf:type rdf:resource="#Pizza"/>
  <hasTopping rdf:resource="#Tomato"/>
  <hasTopping rdf:resource="#Mozzarella"/>
</NamedIndividual>
...

<!-- Define the properties -->
<ObjectProperty rdf:about="#hasTopping">
  <domain rdf:resource="#Pizza"/>
  <range rdf:resource="#Topping"/>
  <minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1<
    /minCardinality>
</ObjectProperty>

<!-- Define the axioms -->
<owl:Class rdf:about="#Pizza">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopping"/>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
        nonNegativeInteger">
```

---

<sup>3</sup>Picture from [19]

Feature	Axiom	Condition	Example
Assertion	$x-y \rightarrow z$	$x-y \rightarrow z$	Leonardo_da_Vinci -painted $\rightarrow$ Mona_Lisa
Same As	$x\text{-sameAs} \rightarrow y$	$x = y$	Mona_Lisa -sameAs $\rightarrow$ The_Mona_Lisa
Different From	$x\text{-diff.from} \rightarrow y$	$x \neq y$	Mona_Lisa -diff.from $\rightarrow$ Mona_Lisa_Cafe

Table 2.1: Main OWL axioms for individuals

```

        1</owl:minCardinality>
    </owl:Restriction>
</owl:equivalentClass>
</owl:Class>

```

There are several famous ontologies that have revolutionized knowledge management and representation in various domains. Some of the most famous ones (some of them named as the respective KG) include:

- DBpedia, a semantic ontology that represents the information contained in the Wikipedia encyclopedia.
- FOAF (Friend of a Friend), used in the field of social networks and the semantic web to describe personal information and relationships between people.
- Cyc, a general ontology used in various applications, containing information about a wide range of concepts and relationships between them.
- GeoNames, used in geography to describe places and their relationships.
- YAGO (Yet Another Great Ontology), a general ontology used to represent domain knowledge that integrates information from various sources, including Wikipedia, WordNet, and GeoNames

### 2.3.2. Axioms

Since OWL is one of the main languages used to write ontologies and offers a much wider range of axioms than RDF, the axioms that can be created with it are examined below.

Along the lines of the axioms outlined by Hogan et al. [15], in table 2.1 the main axioms that can be expressed in OWL concerning individuals are listed.

In the table presented, axioms are formulated in the form of triples. The third column delineates the conditions that must hold between entities for the axiom to be applicable to them. As can be observed from the table, the assertions correspond to the facts typically found within Knowledge Graphs (KGs), the Abox: relationships between individuals or assignments of a class to an individual (e.g., *Leonardo\_da\_Vinci*-type  $\rightarrow$  *Artist*). The "Same As" axiom type is a link

Feature	Axiom	Condition	Example
SubProperty	$x\text{-SubProp.} \rightarrow y$	if $a-x \rightarrow b$ then $a-y \rightarrow b$	IsMotherOf $\text{-SubPr.} \rightarrow$ isParentOf
Domain	$x\text{-Domain} \rightarrow y$	if $a-x \rightarrow b$ then $a\text{-type} \rightarrow y$	Nationality $\text{-Domain} \rightarrow$ Person
Range	$x\text{-Range} \rightarrow y$	if $a-x \rightarrow b$ then $b\text{-type} \rightarrow y$	Nationality $\text{-Range} \rightarrow$ Country
Equivalence	$x\text{-Equiv.Prop} \rightarrow y$	$a-x \rightarrow b$ iff $a-y \rightarrow b$	IsSiblingOf $\text{-Eq.Pr.} \rightarrow$ HasSibling
Inv. Property	$x\text{-Inver.Prop} \rightarrow y$	$a-x \rightarrow b$ iff $b-y \rightarrow a$	IsMotherOf $\text{-Inv.Prop} \rightarrow$ IsChildOf
Transitive	$x\text{-type} \rightarrow \text{Trans.}$	if $a-x \rightarrow b-y \rightarrow c$ then $a-x \rightarrow c$	IsPartOf $\text{-type} \rightarrow$ Transitive
Symmetric	$x\text{-type} \rightarrow \text{Symm.}$	$a-x \rightarrow b$ iff $b-x \rightarrow a$	HasSibling $\text{-type} \rightarrow$ Symmetric
Asymmetric	$x\text{-type} \rightarrow \text{Asymm.}$	if $a-x \rightarrow b$ then not $b-x \rightarrow a$	IsMotherOf $\text{-type} \rightarrow$ Asymmetric
Reflexive	$x\text{-type} \rightarrow \text{Reflex.}$	$a-x \rightarrow a$	Knows $\text{-type} \rightarrow$ Reflexive
Irreflexive	$x\text{-type} \rightarrow \text{Irrefl.}$	not $a-x \rightarrow a$	IsMotherOf $\text{-type} \rightarrow$ Irreflexive
Functional	$x\text{-type} \rightarrow \text{Funct.}$	if $b \leftarrow x-a-x \rightarrow c$ then $b = c$	HasMother $\text{-type} \rightarrow$ Functional
Inv. Functional	$x\text{-type} \rightarrow \text{Inv.Funct.}$	if $b-x \rightarrow a \leftarrow x-c$ then $b = c$	HasEmail $\text{-type} \rightarrow$ IsChildOf
Disjoint	$x\text{-Disj.Prop} \rightarrow y$	if $a-x \rightarrow b$ then not $a-y \rightarrow b$	IsParentOf $\text{-Disj.Prop} \rightarrow$ IsChildOf

Table 2.2: Main OWL axioms for properties

that should exist between equivalent individuals. This particular axiom was previously discussed in paragraph 2.2.2 and proves especially useful for identifying equivalent identities with different namespaces. Conversely, the "Different From" axiom highlights when two individuals represent distinct real-world entities.

Regarding relations, chapter 2.2.2 previously explored how, in the semantic schema, axioms can be established to indicate subproperties, domains, and ranges of a graph's properties. Such axioms are also definable in OWL, among others. Table 2.2 displays the primary axioms related to the properties of a domain of interest.

It should be noted that the "Functional" and "Inverse Functional" axiom types delineate the multiplicity of the relation. They define, respectively, a "many-to-one" and a "one-to-many" relationship. Axioms establishing transitive, symmetric, asymmetric, reflexive, and irreflexive properties indicate specific characteristics of a relation within the domain of interest. Axioms of equivalence, disjunction, and inversion pertain to the relationship of one relation to another.

OWL is characterized by the capability to define a broader range of interrelationships between

Feature	Axiom	Condition	Example
SubClass	$x\text{-SubClass} \rightarrow y$	if $a\text{-type} \rightarrow x$ then $a\text{-type} \rightarrow y$	Country $\text{-SubClass} \rightarrow$ Location
Equivalence	$x\text{-Equiv.Class} \rightarrow y$	$a\text{-type} \rightarrow x$ iff $a\text{-type} \rightarrow y$	Location $\text{-Equiv.Class} \rightarrow$ Place
Disjoint	$x\text{-Disj.Class} \rightarrow y$	if $a\text{-type} \rightarrow x$ then not $a\text{-type} \rightarrow y$	Place $\text{-Disj.Class} \rightarrow$ Person

Table 2.3: Main OWL axioms for classes

classes than what can be defined in semantic schemas. Table 2.3 enumerates the primary ones, which will be essential for understanding the novel approach.

Besides the axioms for subclasses, already present in semantic schemas, in OWL the axioms of equivalence and disjointness between classes are extremely important and will play a crucial role in the new approach that will be presented.

In this study, for clarity, triples  $(h \in V, r \in E, t \in V)$  belonging to a KG  $G=(V,E)$  will be denoted using the notation  $(h,r,t)$ . The axioms of the ontologies, on the other hand, will be represented using the notation found in tables 2.1, 2.2, and 2.3, employing arrows.

### 2.3.3. Ontology based reasoning - Entailment

Given the fact  $(\text{Paris}, \text{type}, \text{Place})$  in the knowledge graph  $G = (V, E)$ , combined with the axiom "Place-EquivalentClass->Location", it can be logically deduced that "Paris-type->Location" is also valid. The significance of the axioms discussed lies not only in the information they convey but also in their potential to derive new, logically consistent information.

To describe how this logical reasoning occurs, some theoretical notions are introduced first.

In the previous chapter, when defining KGs, it was emphasized that there was a correspondence between the nodes and the real-world entities (including classes and categories) of a domain of knowledge, and between the edges and the relationships among these entities. The correlation between the data graph and the real-world entities is so pronounced that, up to this point, the terms "entities"/"relationships" and "nodes"/"edges" have been used interchangeably. For instance, when encountering a node labeled "Mona Lisa", humans can readily associate it with the real-world object.

It is important to highlight that there exists a function mapping nodes and edges of the data graph (represented as URI/IRI) to respective elements in the real world. These latter elements are contained in a graph, the "domain graph", based on real components rather than URI/IRI. It is a graph modeling the domain of knowledge in question, and thus, under the OWA, it might encompass more entities/relationships compared to the nodes/edges of the considered KG. The

"domain graph", along with the mapping function, constitutes the **"interpretation"** of the data graph [4].

An interpretation  $I$  is said to satisfy and be a **"model"** of a graph  $G$  if it adheres to the relationships and properties of the graph. If  $G$  contained only triples from the Abox, i.e., assertions about individuals, for every edge  $(h, r, t)$  in the graph, a corresponding triple would exist in the domain graph of their respective models. However, if the axioms introduced in the previous chapter are added to a graph (e.g., when considering a graph with a schema or a graph alongside its ontology), further conditions are imposed that the interpretations must meet to satisfy the graph. These conditions are precisely those in the 'conditions' column of the tables discussed in the previous section. When considering the Open World Assumption (OWA), meaning more real edges might exist than are present in a KG, the models satisfying a graph are infinite.

Given a graph  $G$  and an ontology  $O$ , the ability to derive or deduce new information from what is already present in the graph and the ontology is termed **"inference"**. This inference process is grounded in the concept of **"entailment"**: the logical implication between a set of propositions and another one. In every inference process, the starting informations from which new information is derived are called "premises" the inferred information "conclusion". The basic principle of deductive logic is that if the premises of an inference are true, also the conclusion must be true. A particular axiom  $A$  is considered "entailed" by  $G \cup O$  if, and only if, it is true for all models of  $G \cup O$  (symbolized as  $G \cup O \models A$ ). In essence, entailment underpins the logical relationship of inference: if  $G \cup O$  entails  $A$ , then  $A$  can be inferred as a logical consequence of  $G \cup O$ . This reasoning applies whether using axioms contained in ontologies or those in semantic schemas and can pertain to axioms between classes and properties as well as assertive axioms involving individuals.

A straightforward example of inference using only axioms related to the T-box is: "Actor-SubClassOf→Person, Person-DisjointClasses→Place" entails "Actor-DisjointClasses→Place."

An example of inference involving also assertive axioms between individuals is: "Tom Cruise-SubClassOf→Actor, Actor-SubClassOf→Person" entails "Tom Cruise-type→Person."

#### 2.3.4. Ontology based reasoning - Consistency

Another pivotal concept, which will be extensively employed in the presented approach, is that of **inconsistency**. A Knowledge Graph  $G$  is said to be inconsistent with respect to an ontology  $O$  when  $G \cup O$  has no models [15]. In such a scenario,  $G \cup O$  is inconsistent. Intuitively,  $G \cup O$  is consistent when there exists at least one interpretation  $I$  that serves as a model for  $G \cup O$ .

An example of a graph inconsistent when combined with an ontology is when the graph includes the triple "Penguin-type->Plant" as well as the triple "Penguin-Type->Animal", and the on-



tology contains the axiom "Plant-DisjointClasses->Animal". If the graph only contained one of these (and all other triples were consistent),  $G \cup O$  would not be inconsistent, even though the triple "Penguin-type->Plant" is illogical based on common knowledge.  $G \cup O$  could also be inconsistent if the ontology itself, on its own, contained contrasting axioms. For instance, if an ontology contains the axioms "Place-EquivalentClass->Person" and "Place-DisjointClasses->Person", there can never be an interpretation  $I$  that satisfies both. Based on basic logic rules, the first axiom indicates that the set of entities of type "Place" is equivalent to that of "Person", while the second axiom asserts that these two sets cannot share common elements. This contradiction renders the ontology containing the two axioms intrinsically inconsistent.

Today, the processes of inferring new axioms and checking consistency can be performed automatically with the assistance of specialized software known as ontological reasoners. These tools utilize logical rules to deduce new axioms. Moreover, consistency checks can also consider axioms deduced from the initial ones. Examples of such reasoners include:

- **HermiT**: Suitable for complex OWL ontologies but can be slow for very large ontologies.
- **Pellet**: In addition to reasoning capabilities, it provides detailed explanations of inconsistencies but demands more hardware resources compared to alternatives.
- **Jena**: Comprises an entire framework for handling RDF, RDFS, OWL, and executing queries, but is less optimized for complex ontologies.

Each one of these could be more appropriate in specific scenarios as they have different strengths and weaknesses.

## 2.4. KG Refinement

Knowledge Graphs are highly powerful tools for organizing and representing knowledge. However, they do have limitations that can hinder their effectiveness in certain contexts. These limitations include the fact that their completeness and correctness is highly dependent on the technique used to create them, as the process to identify all existing relationships between entities is not trivial to do in a complete and accurate manner. Additionally, there are currently not many algorithms that allow for direct processing of information using KGs. The utilization of Knowledge Graph embeddings (KGE) , that will be presented in paragraph 2.5, can help to overcome the latter problem. Regarding completeness and correctness, there are machine learning methods specifically designed to address these issues.

### 2.4.1. Missing edges in knowledge graphs

Given a knowledge graph  $K$ , a triple  $(h, r, t)$  is defined as "**positive**" if there exists an edge labeled  $r$  connecting node  $h$  to node  $t$ , that is  $(h, r, t) \in K$ . Otherwise, the triple is defined as

"negative".

The edges in a Knowledge Graph represent known relationships. However, KGs are not always complete, as there may be missing edges where entities are related but the relationship is not represented in the graph (these triples are still considered as negative). There can be several reasons for missing edges in a KG, including:

- incomplete data sources from which the graph was constructed (possibly due to the type of representation used in these sources, which may not capture all aspects of the entities);
- errors/inaccuracies in the data leading to missing relationships during the extraction/integration/processing steps done during the creation of KGs;
- limitations of the formalism used in the graph not suitable to represent more complex relationships or those regarding a different domain;

Missing edges in a KG can be interpreted differently depending on the reasoning approach used:

- Under the Closed World Assumption (CWA), it is assumed that all triples not present in a graph are false.
- The Open World Assumption (OWA) assumes that everything not known to be false could potentially be true: missing edges are simply unknown and cannot be assumed about. This is the general assumption made when talking about knowledge graphs.
- Another approach is to use the Local Closed World Assumption (LCWA), a variant of CWA. LCWA assumes that everything not known to be true or false is unknown within a given context. In a KG  $G=(V,E)$ , only triples of which there is local knowledge, i.e., those involving a particular entity in question  $e \in E$  and that are known to be true are considered true. The assumption does not apply to facts involving other entities. More formally, in a KG  $G=(V,E)$  given an entity  $e \in E$ , only triples involving 'e' that are already known are considered true.

Having missing edges in a KG is problematic as it can affect the accuracy and completeness of the represented knowledge. This can lead to undesired effects and/or limit its usefulness in various application domains.

There are different approaches that can be used to minimize this problem, such as techniques for completing or predicting missing edges in KGs. These include rule-based methods, statistical models, and link prediction algorithms that leverage the information already present in the KG to predict missing triples. Other solutions involve using external data sources (such as ontologies or other knowledge graphs) to integrate and validate the information contained in the KG.

### 2.4.2. Completion - Link prediction

The link prediction algorithms use the graph itself to enhance its completeness. These algorithms predict the existence of a link between two entities in a knowledge graph by computing the likelihood of the positivity of missing edges in the graph. This allows for the identification of potential relationships that might not have been captured during the graph's creation phase. For instance, consider a KG containing the following facts: "Mario Rossi-LivesIn→ Milan", "Mario Rossi-MarriedTo→ Carla Verdi", "Carla Verdi-WorksIn→ Milan", "Mario Rossi-HasChild→ Federico Rossi", "Federico Rossi-StudiesIn→ Milan". A likely high-probability link might be "Mario Rossi-livesIn→Milan".

There are various implementation techniques for link prediction algorithms, ranging from neural networks, machine learning algorithms, to algorithms that leverage vector representations of a graph's entities, known as embeddings (presented in section 2.5).


A task similar to link prediction but conceptually different is "triple classification." Here, the focus is on the validity of each individual triple. As with any other classification task, based on a threshold, the triple will be classified as likely positive or not. The triple classification algorithm can also be used to verify that triples predicted as probable in "link prediction" are also classified as positive.

### 2.4.3. Correction - Fact validation and inconsistency repair

Knowledge graphs (KGs) are susceptible to noise: they might contain relationships between entities that are not true within the domain of knowledge. It is crucial to emphasize that such triples are still defined as positive since they belong to the KG. Correcting a knowledge graph involves identifying and removing these incorrect triples. The two primary tasks for correction are "fact validation" and "inconsistency repair."

During the "fact validation" process, a "plausibility" score is assigned to each triple in the graph. This score is calculated based on the properties of the entities within the graph itself or by referencing external sources.

Inconsistency repair, on the other hand, is performed using ontology axioms, such as those of disjunction. As highlighted in paragraph 2.3.4, when considering a graph along with its ontology, it may turn out to be inconsistent. Inconsistencies in KGs compromise their reliability, accuracy and lead to potential misinformation. Therefore, their removal is a crucial process. Inconsistency repair techniques involve, as a first step, identifying the triples that are the root cause of the inconsistency. For instance, reverting to the example mentioned in chapter 2.3.4, the inconsistent triples identified that caused an inconsistency were <Penguin, type, Plant> and <Penguin, type, Animal>. Once inconsistent triples are identified, various solutions have been proposed on how to decide which triples to remove. Topper et al. [29] suggest a "manual" removal approach. In



ID	Color
1	Blue
2	Green
3	Red
4	Red
5	Green

ID	Color_Blue	Color_Green	Color_Red
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

Figure 2.9: Example of 1-hot encoding technique.

the case of very large KGs or those with numerous inconsistencies, this can be a time-consuming process. An automated method has been proposed by Bonatti et al. [6], which suggests deciding based on the reliability of the sources of facts and on how many inconsistencies a particular triple might cause.

## 2.5. Knowledge graph embeddings

### 2.5.1. Embeddings basics

Embeddings are fundamental elements in modern Machine Learning techniques, as they allow for the processing of non-numeric data in algorithms designed exclusively for numeric data.

In the realm of Machine Learning, an embedding is defined as a low-dimensional space where high-dimensional vectors can be translated [31]. Non-numeric data, such as words or phrases, can be represented with high-dimensional and highly sparse vectors. Through embeddings, these can be mapped to low-dimensional vectors, simplifying their processing. Moreover, embeddings also encode the semantics of the variables they represent: semantically similar data are represented by embeddings that are close in the low-dimensional space (or embedding space).

Embeddings are particularly crucial in branches of Machine Learning where non-numeric data play a key role, such as Natural Language Processing (NLP), Computer Vision, and recommendation systems.

Before the advent of embeddings, categorical variables in ML algorithms were represented using one-hot encoding: each categorical variable was mapped to a number of binary features equal to the number of categories. These features would contain a '1' if the data in question had the variable matching the category indicated by the column and '0' otherwise. Figure 2.9 provides a simple illustration of one-hot encoding.

The issue with one-hot encoding becomes evident when there are many categories for a variable (high-dimensional), leading to the creation of numerous sparse vectors, predominantly composed of zeros. Another point to highlight is that the binary vectors generated entirely omit the semantic aspect of the input data. For instance, if the categorical variable were "food", the

category "apple" would be equally different from "banana" and "bread" in a one-hot encoded representation.

With embeddings, both these challenges are addressed: the low-dimensional vector space they reside in quantifies the semantic similarity between categories. This implies that semantic similarity can be measured as similarity between embedding vectors, measured using metrics such as cosine similarity or Euclidean distance. On the other hand, their computation is more complex compared to one-hot encoding, and the vectors themselves are less straightforward to interpret. Embeddings are currently utilized in numerous fields and for a wide range of tasks, including:

- **Text Analysis.** By capturing the semantic meaning of a word, embeddings play a pivotal role when analyzing texts. In this context, they are used for various tasks such as: search in texts based on the meaning of words, not just the keyword matches, the computation of similarity between texts, the classification of texts based on their meaning or the subjects they address.
- **Recommendation systems.** The primary goal of recommendation systems is to suggest content that users may find interesting. They are employed across numerous platforms, from e-commerce to streaming services and social media. Embeddings play a vital role here, representing the content to suggest. For instance, in an e-commerce site, two similar products could have analogous embeddings. If a user expresses interest in one, the other is likely to be suggested. In recommendation systems, embeddings can also model the users themselves, reflecting their preferences, past purchases, and searches. This helps creating better tailored recommendations.
- **Community detection:** By leveraging the similarity between embeddings, traditional clustering algorithms like K-means or DBScan can be applied, grouping embeddings into distinct communities.
- **NLP tasks:** In NLP, embeddings play a pivotal role, aiding in identifying named entities (NER), determining parts of speech, understanding user queries for relevant responses in question answering, and generating coherent sentences or predicting subsequent words in language modeling."
- **Representation of images, audio, and abstract concepts:** Embeddings can also represent such intricate data types. For images, they may capture intricate visual characteristics, aiding tasks like object recognition or new image generation. Regarding audio, they might embed features like word presence, timbre, or other characteristics differentiating various musical genres. Lastly, they can depict abstract concepts like artistic genres or film themes, capturing the nuances of such notions.

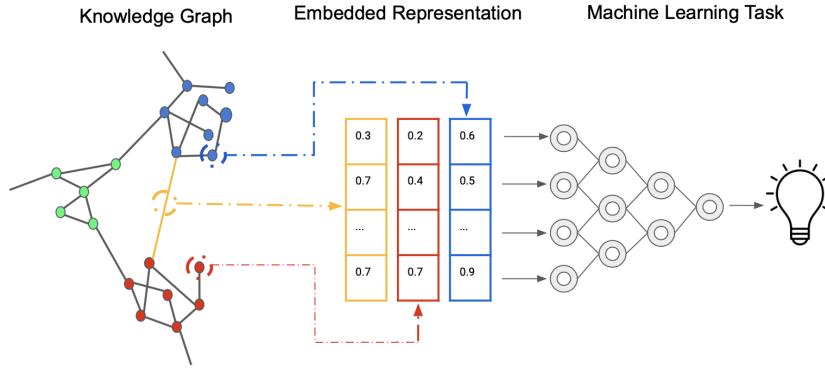


Figure 2.10: Embedding of a knowledge graph for a machine learning task.

### 2.5.2. Knowledge graph embeddings basics

After delving into the concept of embeddings in general, the focus now shifts to knowledge graph embeddings. As previously mentioned in chapter 2.5.1, there are currently limited algorithms available for the direct processing of information using KGs. Knowledge graph embeddings, which provide a more dense and efficient representation of the graphs, can help to overcome these limitations.

Knowledge Graph embeddings (KGEs) are supervised learning models that map entities and relations of a knowledge graph into low-dimensional vectors, the embeddings, while capturing their semantic meanings [17]. Each entity and relation is translated into a vector of dimension  $d$ , the "embedding dimension", which can generally differ between vectors that represent entities and those that represent relations. In general, vector representations of real numbers are used, but there are methods that use matrices to represent relationships and complex vectors to represent entities and relationships.

The embeddings are generated in such a way to capture latent properties of the semantics in the knowledge graph: similar entities and similar relationships will be represented with similar vectors [5].

In general, by using embeddings of a KG, it is possible to represent all semantic information efficiently and compactly. In this way, node embeddings become a fundamental tool for fully exploiting the information contained in knowledge graphs, making them a valuable resource for machine learning and artificial intelligence applications (presented in 2.10).

### 2.5.3. Embedding models basics

Today, there exists a plethora of models to create KGEs. The majority of them differ based on diversity in these 3 key points [17]:

1. the choice of the representation space

2. the scoring function used
3. the loss function that characterizes them

The representation space is the multi-dimensional space in which the vector/matrices of entities and relations will lie.

The scoring function  $f(h, r, t)$  takes as input the triple  $(h \in V, r \in E, t \in V)$  of a graph  $G = (V, E)$ . This function is used to assign a score to each triple, representing its plausibility or the likelihood of it being true. This scoring is crucial during training process to learn embeddings. The goal is to optimize the embedding vectors of  $h, r$ , and  $t$  so that existing triples in the graph have a certain range of scores, while non-existing triples have another range of scores. This optimization is achieved through the minimization of the loss function. Consequently, the plausibility of a triple  $(h, r, t)$  is measured with the score produced by the function  $f(h, r, t)$ . The model learned by the training algorithm is the embeddings themselves.

The training requires both positive and negative triples: the quality of the negative triples generated is crucial for KGE models, because if the negative triples are not representative of all the relations and entities of the KG, the model will not be able to correctly distinguish between new positive and negative new triples, in other words it will not be able to generalize well on unseen data. This could lead to a loss of accuracy in predictions and therefore a reduction in the overall performance of the model. Another aspect to consider to ensure that the model learn the embeddings of the KG in a comprehensive way, is the balance of the number of positive and negative samples used during the training. An unbalanced dataset can lead the model to be biased towards the class with more samples. The creation of negative triples is conditioned by the choice of the reasoning approach (CWA/LCWA/OWA). Indeed, using CWA, all the missing triples in the graph are considered as negative, and this can lead to wrong conclusions in case they are actually true (false negative triples). In the case of OWA, generating negative triples in this scenario can be problematic as it is not possible to be entirely sure that they are indeed negative unless there is specific information about the absence of a particular relation between two elements. Finally, in the case of LCWA, negative triples can be created by generating triples that do not appear in the graph within the considered context.

The way of generating these triples, together with the scoring and loss functions, are aspects that strictly depend on the type of KGE model chosen. In Chapter 3, three algorithms will be presented, which will serve as a basis for the training model used in the new presented pipeline.

The pseudocode presented in Algorithm 2.1, from Yuanfei et al. [9], represents the main steps of the majority of processes to train Knowledge Graph Embedding models.

Specifically, the positive triples used for training are typically a subset of the triples from the KG, usually comprising 70-80% of them. The remaining triples are reserved for testing and validation

---

**Algorithm 2.1** Generic Training Algorithm for KGE models

---

- 1: **Input:** training set  $S = \{(h, r, t)\}$ , entity set  $E$ , relation set  $R$ , embedding dimension  $k$
  - 2: **Output:** entities and relations embeddings
  - 3: **Initialization:** Embeddings are randomly initialized
  - 4: **while** non-stop condition **do**
  - 5:    $S_{\text{batch}} \leftarrow \text{sample}(S, b)$  {Randomly sample a batch of size  $b$  from the training set}
  - 6:   Create a set of negative, corrupted triples  $S_{\text{corrupted}}$
  - 7:    $T_{\text{batch}} \leftarrow T_{\text{batch}} \cup S_{\text{batch}} \cup S_{\text{corrupted}}$  {Add positive and negative triples to the training set}
  - 8:   Update embeddings by minimizing the loss function
  - 9: **end while**
- 

phases. During this partitioning, it is essential to ensure that all entities are represented in the training set.

The testing phase is used to establish the quality of embeddings, which reflects their ability to capture the semantic relations between entities in a KG such as hierarchies, properties, and relationships between them, can be evaluated with different methods. Among the main techniques, the tasks of "link prediction" and "triple classification", presented in paragraph 2.4.2, can be utilized for this purpose. Specifically, link prediction can leverage the embeddings of entities to compute the similarity between them, using this information to determine the likelihood that a triple (in the testing set) is true or false. Similarly, triple classification can use the obtained embeddings to compute a score for various test triples. Based on this score, and by setting a threshold as in all classification algorithms, a triple is classified as positive or negative.



## 3 | Related Work

Various models and techniques have been proposed in the literature for training embeddings of knowledge graphs. A large part of them differs from other solutions in the way negative triples are created [21][25][11][2][27]. In this section, the focus is on algorithms that produce embeddings in  $\mathbb{R}$ . Specifically, two foundational algorithms, which underpin the training algorithm utilized in the newly introduced approach, are presented. Furthermore, in sections 3.2 and 3.3 two significant works that have greatly contributed to the development and conceptualization of the new approach are discussed.

### 3.1. Two translational models: TransE and TransOWL

Translational models represent a primary category of Knowledge Graph Embedding models. Within these models, the score function is designed such that positive triples yield a score  $\approx 0$ , while negative ones produce a score different from 0. The underlying premise is to depict the semantic relationships between entities in terms of geometric distances within the embedding space. The intuitive foundation of this score function is that, in the embedding space, entities linked by relationships should be proximate. Consequently, by summing the embeddings corresponding to the head and the relationship of a triple, one should arrive at an embedding similar to that of the tail if the triple is positive. Thus, the objective of embedding models is to ensure that each positive triple has a score (in absolute value) lower than all its negative counterparts. It becomes evident that the efficacy of the embedding models is intrinsically tied to the quality of the generated negative triples. Furthermore, given that Knowledge Graphs exclusively store positive triples, producing genuine real negative triples emerges as a challenging problem.

#### TransE

The pioneering model in this domain is known as the Translating Embedding, or TransE [7]. This simple model produces embeddings within  $\mathbb{R}^k$ , where  $k$  serves as the model's hyperparameter. For clarity, the embedding of an entity or relation  $x$  is denoted as  $e_x$ . The underlying principle of TransE is that, as in the other translational models, the score of positive triplets should be approximately zero, given  $e_h + e_r \approx e_t$ . Here,  $e_h$ ,  $e_r$ , and  $e_t$  represent the embedding vectors of the head  $h$ , the relation  $r$ , and the tail  $t$  of a triplet  $(h, r, t)$ , respectively. On the contrary, when

a triplet  $(h, r, t)$  is negative, the sum  $e_h + e_r$  should be distant from  $e_t$ . The scoring function is defined as:

$$f(e_h, e_r, e_t) = \|e_h + e_r - e_t\|_2^2 \quad (3.1)$$

The loss function to minimize is defined as:

$$L = \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) \quad (3.2)$$

In the above exposition,  $S$  represents the set of positive training triplets, while  $S'$  denotes the set of negative triples. These negative triplets are specifically generated from the positive ones: for each positive triplet  $(h, r, t)$ , a negative triplet  $(h', r, t')$  is randomly generated. To be precise, given a positive triplet  $(h, r, t)$ , the more accurate representation of the negative triplet would be either  $(h', r, t)$  or  $(h, r, t')$ . This is because the negative triplet is produced by replacing either the head  $h$  or the tail  $t$  with another entity randomly selected from the graph, represented as  $h'$  or  $t'$ , respectively. Formally:

$$S'(h, r, t) = \{(h', r, t) | h' \in E\} \cup \{(h, r, t') | t' \in E\} \quad (3.3)$$

This hyperparameter is pivotal because it aids the model in differentiating positive triplets from negative ones more effectively.

As observed, by minimizing  $L$ , the score conferred to the positive triplet is reduced, and the score for the negative triplet is incremented. Specifically, the minimization of  $L$  is achieved using the Stochastic Gradient Descent (SGD) optimization method. This method is widely embraced in machine learning model training. It dictates that the parameters of a model (in this context, the embeddings themselves) are updated in the direction opposite to the gradient of the partial derivatives of the loss function. Formally:

$$e_{\text{new}} = e_{\text{old}} - \alpha \times \nabla_{e_{\text{old}}} L \quad (3.4)$$

The parameter  $\alpha$  dictates the magnitude of the updates: how much each adjustment impacts the current value of the embedding.

In 3.1, one can find a concise pseudocode delineating the functionality of TransE. A noteworthy aspect is that for each pair of positive and negative triplets created during the training, the associated embeddings undergo an update. More specifically, the embeddings are updated whenever the chosen triplet pair results in a "loss" that contributes to one of the summation terms of the overall loss function. Thus, every time that, given a positive triplet  $e_h, e_r, e_t$  and a negative triplet  $e_{h'}, e_r, e_{t'}$  the following condition holds:

---

**Algorithm 3.1** TransE pseudocode

---

```

1: Initialize embeddings randomly
2: for  $i = 1$  to trainTimes do
3:   for  $j = 1$  to size(KG) do
4:     Sample positive triple  $(h, r, t)$  from  $S$ 
5:     if random_number(0,100) < 50 then
6:       Choose entity  $j$  randomly
7:       Negative triple  $(h', r, t') = (j, r, t)$ 
8:     else
9:       Choose entity  $j$  randomly
10:      Negative triple  $(h', r, t') = (h, r, j)$ 
11:    end if
12:    updateEmbeddings( $h, r, t, h', r, t'$ )
13:  end for
14: end for

```

---

$$L = \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) > 0 \quad (3.5)$$

an update will be executed on the involved entities and relations, specifically  $e_h, e_r, e_t, e_{h'}, e_{t'}$ .

Additionally, during the training phase, a constraint is enforced: the L2-norm of the embeddings is set to 1. In other words,  $\|e\|_2 = 1$  for all generated embeddings. Embeddings during the training can grow unbounded in magnitude without this constraint, leading to a very good performance on the training data but bad generalization capabilities on unseen data.

In the TransE model, negative triples are generated randomly. The sole constraint applied in practical implementations of the algorithm ensures that the negative triples do not belong to the training set. Beyond this, no further checks are implemented to verify that the generated triple is indeed incorrect and this leaves room for the possibility of creating triple "false negative". This limitation has led to the development of models such as TransOWL, which aim to produce negative triples that are genuinely incorrect.

### TransOWL

In TransOWL, created by the authors of "Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs"[10], the problem of generating negatives is partially solved by taking as negative examples pre-computed triples that are inconsistent with the Knowledge Graph and its associated ontology. Several studies have advocated the use of ontologies for deriving embeddings: the authors of OWL2VEC [8] and Schockaert et al.[13] introduce a method to generate embeddings of the ontologies themselves. The authors of EmbedS incorporate ontological knowledge into the embedding creation process, leveraging a geometrical interpretation of triples and axioms. Furthermore, Serrano et al. [3] put forth a methodology to harness ontologies during the embedding training phase, with a specific emphasis

on entities not present in the training set.

The underlying idea of TransOWL is that the background knowledge provided by the axioms within the semantic schema and ontologies can be leveraged to create negative triples that are genuinely false. By doing so, the embeddings produced can encapsulate implicit information regarding these axioms, not merely the Abox present in the graph. TransOWL may be seen as version of TransE augmented with background knowledge. The updating procedure, using Stochastic Gradient Descent, is exactly the same as in TransE (function "*updateEmbeddings*" in pseudocodes 3.1 and 3.2). During the training phase, the axioms are not only employed for generating negative triples but also for adding new positive ones.

In 3.2, the pseudocode of TransOWL is analyzed. Examining it is crucial because, as will be seen in subsequent sections, the training used in the new approach will also utilize the axioms from ontologies for the generation of negative triples, and the differences compared to TransOWL will be highlighted.

The Equivalent Classes axiom is leveraged to construct an additional positive triple, to which another random negative triple will be associated. The Inversion axiom, on the other hand, modifies the embedding of inverse relationships as if, starting from the positive triple  $(h, r, t)$ , two relationships between  $h$  and  $t$  were considered simultaneously: one  $r$  and an inverse to  $r$  with an opposite sense. Similarly, with the Equivalence Axiom, both  $r$  and its equivalent are considered between  $h$  and  $t$  at the same time. The Subclass Axiom is used to establish an additional constraint that the inventors of TransOWL [10] define as expressing "major specificity". This means that given a positive triple  $(h, r, t)$  and a subclass of  $t$ ,  $t'$ , the positive triple  $(h, r, t')$  should have a score closer to 0 as it expresses a fact more specific than the former.

Given the structure of the algorithm, it becomes easy to comprehend the structure of the loss function that TransOWL aims to minimize, as shown in 3.6. In particular, the score function is precisely that of TransE, referred to in formula 3.1. The various terms of the loss are indicated in different summations based on the additional updates made compared to TransE, due to the presence of various axioms: *equivalentClass*, *equivalentProperty*, *inverseOf* and *subClassOf*.

$$\begin{aligned}
L = & \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'} \max(0, \gamma + f_r(e_h, e_t) - f_r(e_{h'}, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{inverseOf}}} \sum_{(h',r,t') \in S'_{\text{inverseOf}}} \max(0, \gamma + f_r(e_h, e_t) - f_r(e_{h'}, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{equivProperty}}} \sum_{(h',r,t') \in S'_{\text{equivProperty}}} \max(0, \gamma + f_r(e_h, e_t) - f_r(e_{h'}, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{equivClass}}} \sum_{(h',r,t') \in S'_{\text{equivClass}}} \max(0, \gamma + f_r(e_h, e_t) - f_r(e_{h'}, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{subClass}}} \sum_{(h',r,t') \in S'_{\text{subClass}}} \max(0, (\gamma - \beta) + f_r(e_h, e_t) - f_r(e_{h'}, e_{t'}))
\end{aligned} \tag{3.6}$$

---

**Algorithm 3.2** TransOWL pseudocode

---

```

1: Initialize embeddings randomly
2: for  $i = 1$  to trainTimes do
3:   for  $j = 1$  to size(KG) do
4:     Sample positive triple  $(h, r, t)$  from  $S$ 
5:     if random probability  $< 0.5$  then
6:       Choose entity  $j$  outside relation  $r$  domain or random. Neg. triple  $(h', r, t') = (j, r, t)$ 
7:     else
8:       Choose entity  $j$  outside relation  $r$  range or random. Neg. triple  $(h', r, t') = (h, r, j)$ 
9:     end if
10:    if  $r$  is "typeOf" then
11:       $updateEmbeddings(h, r, t, h', r, t')$ 
12:      if tail  $t$  has equivalent classes then
13:        for each equivalent class do
14:          Generate positive triple  $(h, r, equivalentClass)$ 
15:          Generate negative triple as in TransE
16:           $updateEmbeddings(h, r, t, h', r, t')$ 
17:        end for
18:      end if
19:    else if relation  $r$  has an inverse then
20:       $updateEmbeddings(h, r, t, h', r, t') +$  update the embedding of the inverse
21:    else if relation  $r$  has an equivalent then
22:       $updateEmbeddings(h, r, t, h', r, t') +$  update the embedding of the equivalent
23:    else
24:       $updateEmbeddings(h, r, t, h', r, t')$ 
25:    end if
26:    if relation  $r$  is "typeOf" and tail  $t$  has subclasses then
27:      for each subclass  $t'$  of  $t$  do
28:        Update embeddings such that  $f(h, r, t) > f(h, r, t')$ 
29:      end for
30:    end if
31:  end for
32: end for

```

---

Where:

- $S_i$ , with  $i \in \{\text{inverseOf}, \text{equivProperty}, \text{equivClass}, \text{subClass}\}$ : represents the additional updates done thanks to the axioms.
- $\gamma$ : is the margin hyperparameter, serving the same function as in TransE.
- $\beta$ : is a factor required to determine the direction of the inequality concerning the specificity of subclasses.

Special attention should be paid to the fact that this algorithm uses a significant portion of triples that are generated randomly. This corresponds to the first summation that appears in the formulation of the loss function 3.6. As can be seen in pseudocode 3.2, at lines 6, 8, and 15,

the creation of negatives occurs by corrupting either the head or the tail in a random manner, as in TransE.

Retaining a subset of triples generated randomly proves advantageous in the embedding creation process. By adopting this strategy, the model can access a wide array of entities, ensuring uniform learning across the knowledge graph and bolstering the model's generalization capabilities. Thus helping models in accurately representing the real-world knowledge graph and its capability in predicting new facts not encountered in the training set.

The TransOWL algorithm has served as a foundational basis for training the novel approach that will be introduced in the subsequent sections.

### 3.2. Graph structure for negative triples creation

The challenge of generating negative samples during training is a widely debated issue, and various solutions have been proposed to address it. As discussed in the preceding paragraph, TransE creates negative samples by randomly corrupting the head or tail of each triple. In contrast, TransOWL relies on ontological axioms when they are available. Some techniques do not resort to external data sources as ontologies but leverage the inherent properties of the knowledge graph itself to create meaningful negative triples for learning purposes [24][22].

#### Exploring sparse regions of the graph

In their paper "Towards Loosely-Coupling Knowledge Graph Embeddings and Ontology-based Reasoning," Kaoudi et al. [18] not only integrate ontological knowledge in the creation of triples but also introduce a component designed to identify triples that, although not present in the graph, could likely be true. Such triples should therefore be avoided as negative samples. These high-probability true triples are sought by exploring sparse regions of the graph, under the premise that "entities that are densely connected are less probable to have missing true relations." Sparse regions are pinpointed using density metrics, such as the clustering coefficient of entities. Once potential positive triples in these sparse regions are identified, they are consciously avoided as negative samples during training.

#### Exploring entities neighborhoods

Within the context of utilizing the graph's structure for negative sample generation, another notable contribution comes from Ahrabian et al. [1]. In their work titled "Structure Aware Negative Sampling in Knowledge Graphs," they exploit the graph's structure to generate hard negative samples. These samples embed critical data characteristics that are typically challenging to learn. Specifically, in this paper, the entities to be corrupted from each positive triple (either head or tail) are replaced by entities within their k-hop neighborhood, where k serves as a hyperparameter. A graphical representation of this approach can be seen in Figure 3.1.

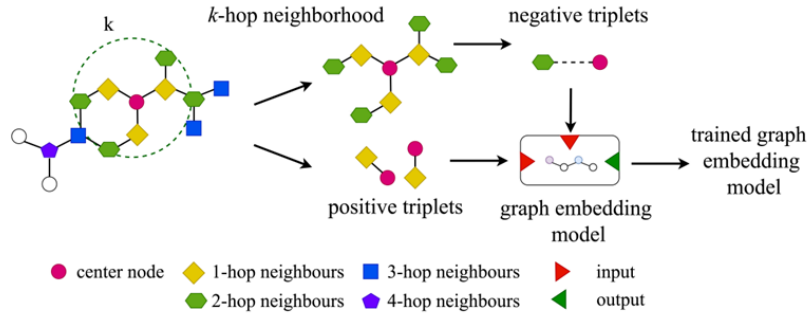


Figure 3.1: Graphical representation of the SANS approach.

This approach, dubbed by the authors as SANS (Structure Aware Negative Sampling), hinges on the premise that entities within the neighborhood of a given entity are more likely to be correlated with it, making them more challenging to distinguish from it. The authors assert: "we argue that such local negatives are harder to distinguish and lead to higher scores as evaluated by the embedding model". Specifically, the authors emphasize that as ' $k$ ' assumes a smaller value, the negatively generated triples become increasingly meaningful, thus enhancing the learning process. They note: "the semantic meaningfulness of SANS based negatives decline as we increase the size of the  $k$ -hop neighbourhood". The efficacy of the SANS approach has been demonstrated on standard knowledge graphs, namely FB15K-237, WN18, and WN18RR, with experiments conducted using neighborhood radius values ( $k$ ) ranging from 2 to 8.

When corrupting an entity, to ascertain whether a potential substitute is part of the original entity's neighborhood, one must explicitly construct the neighborhood. This can be a computationally expensive operation, both in terms of time and memory, especially for large and/or dense datasets. To address this challenge in such scenarios, the authors introduce a variant of the SANS method called RW-SANS. This approach suggests approximating an entity's neighborhood by performing random walks of length  $k$  originating from the entity in question. While this results in an approximated neighborhood rather than an exact one, the authors have shown that this approximation does not compromise the method's effectiveness.

It is noteworthy that in this implementation, there remains a component of negative triples generated by randomly corrupting either the head or tail of positive triples. This is because not all entities possess  $k$ -hop neighbors, and when they don't, the corruption occurs at random. Moreover, in the RW-SANS approach, if the random walks fail to achieve a length of  $k$  due to encountering a node without outgoing edges, a neighbor is added through random sampling.

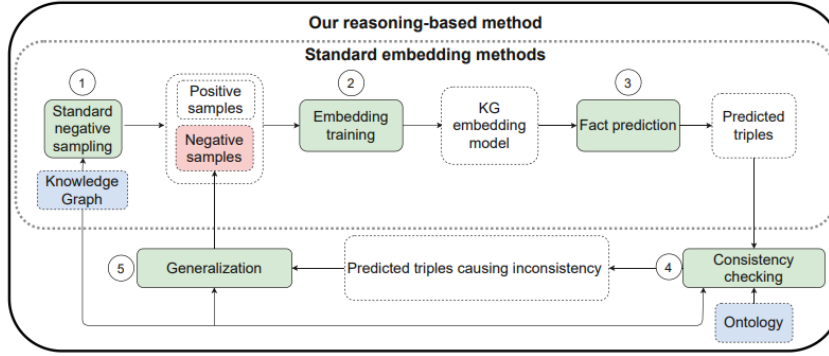


Figure 3.2: Pipeline of iterative ontology-based reasoning.

### 3.3. Iterative approach for negative sampling

The iterative training of embeddings has been proposed by various authors in the literature [34][35][28]. In this section, the approach most akin to the one employed in this study is introduced. In the paper "Improving Knowledge Graph Embeddings with Ontological Reasoning", Jain et al. [16] propose an iterative framework that leverages ontological reasoning to enhance the quality of the produced embeddings. Specifically, their framework encompasses four main steps:

1. **Training:** The training is conducted using well-established algorithms such as TransE.
2. **Link Prediction:** The embeddings generated during the training phase are employed in a link prediction algorithm to forecast potential new connections between entities. At the conclusion of this step, a set of links predicted by the model, which were not part of the training data, will be obtained.
3. **Consistency Checking:** During this step, the set of new triples produced through link prediction is scanned, and only those triples that induce inconsistencies when added to the original graph are extracted.
4. **Generalization Component:** Once the inconsistent triples predicted by the model are identified, their count is augmented through a "generalization" process. This process generates similar triples from each identified triple by replacing either the head or the tail with entities having a similar neighborhood. These triples are then utilized as new negatives in a subsequent training iteration.

The schematic representation of this framework, as proposed by the authors of [16], is depicted in Figure 3.2<sup>1</sup>.

<sup>1</sup>Picture from [16]



From this, it is possible to observe that in this instance, ontologies are not employed during the training phase. Instead, they function externally to the training, aiming to identify triples that the model predicts as positive but should be negative due to their inconsistency. Thus, these are used to "correct" the relative embeddings in the next training by being used as new negative triples. Another study that discuss methods to exploit embeddings in the negative creation is the one proposed by Kotnis et al.[20].



## 4 | The new approach

Generating high-quality KG embeddings remains a challenge in contemporary research, and various methods have been proposed in the literature for their creation. This section introduces a novel method for producing quality embeddings, TransHI. This approach aims to address issues found in state-of-the-art methods and enhance the quality of the generated embeddings. It comprises an iterative pipeline designed to enhance the quality of embeddings through extensive use of ontologies, of the structure of KGs and other heuristic strategies. Initially, the primary contributions and characteristics of the new method are outlined. Subsequently, a general overview of the iterative structure and the roles of its various steps is presented. This is followed by a detailed explanation of each step within the pipeline.

### 4.1. TransHI - The main contributions

The proposed method is TransHI, an iterative technique that employs a hybrid approach for the generation of Translational embeddings. Unlike the algorithms TransE, TransOWL, and SANS introduced in Section 3, this method extends beyond merely the training phase. It includes an initial phase where the KG is preprocessed and a procedure to conduct the embedding training multiple times in an iterative manner. Specifically, the "hybrid" aspect of this approach pertains to the first iteration of the embeddings training: negative triples are generated in two ways that leverage different facets of the KGs. They are crafted both by utilizing axioms extracted from the ontologies associated with the KGs and by harnessing the structural properties of the KGs themselves.

As will be discussed in the following section, the method consists of several steps. However, these can be executed in a fully automated manner, without human intervention between the various stages.

The primary challenges inherent in state-of-the-art approaches that TransHI aims to address include:

- **Presence of inconsistencies within KGs.** As previously discussed in Section 2.4.3, inconsistencies within KGs compromise their accuracy and reliability. While there are studies in the literature dedicated to refining KGs [29][6][32], the removal of inconsistencies is typically not incorporated into methodologies used for embedding creation. TransHI

employs a reasoner to pinpoint the triples in KGs that lead to inconsistencies and adopts an effective heuristic method to determine which ones to remove.

- Difficulty in generating uniform negative training triples.** TransOWL, introduced in Section 3.1, while an improvement over TransE because it removes the probability of creating "false negative" triples, struggles to construct effective negatives using all the relationships and entities of the KG. This is primarily because a KG's ontology typically provides information on only a subset of possible relationships and entities. TransHI addresses this limitation of TransOWL by leveraging its hybrid approach. Triples generated from the ontology are complemented by those derived from the graph's structure, ensuring that triples can be created for every relationship and entity. This dual strategy ensures not only a uniform coverage of entity and relationship sets in the training set but also that the embeddings encapsulate both the semantic information extractable from a KG's ontology and insights from its structure.
- Challenges in harnessing the semantic richness of ontologies.** TransOWL only utilizes axioms directly extractable from the ontology, neglecting the inferable ones. As will be elaborated upon, TransHI incorporates a dedicated step in the pipeline for inferring a set of axioms that can be inferred from an ontology, ensuring that a richer semantic content of the ontology is utilized in the following phases of the approach.
- Creation of numerous and effective training triples for subsequent iterations.** As presented in Section 3.3, Jain et al. [16] suggest to check, after producing a set of embeddings, if a link prediction algorithm predicts as probable triples that are in reality inconsistent with the KG. If such triples exist, the recommendation is to use them as negative samples in subsequent training iterations to refine the corresponding embeddings. However, their method bears a significant drawback. By leveraging link prediction, which inherently aims to identify potential true triples rather than focusing on inconsistent ones, the number of inconsistent triples predicted may be quite limited. TransHI addresses this shortcoming. Specifically, after generating embeddings using a training algorithm, a pool of triples inconsistent with the KG is formed solely based on the training set and the ontology. Subsequently, a classification task, employing the produced embeddings, discerns these inconsistent triples as either positive or negative. Given that these triples are inherently inconsistent, they should all be classified as negative. Those misclassified can then be employed as negative samples in a suggested subsequent iteration to refine the respective embeddings. By centering its attention exclusively on inconsistent triples, this mechanism identifies a significantly larger number than a typical link prediction algorithm. As a matter of fact, in this scenario, one can dictate the input data, whereas, with link prediction, the selection process for likely links is automatic. Moreover, the classification algorithm itself is considerably less computationally intensive than the link prediction one.

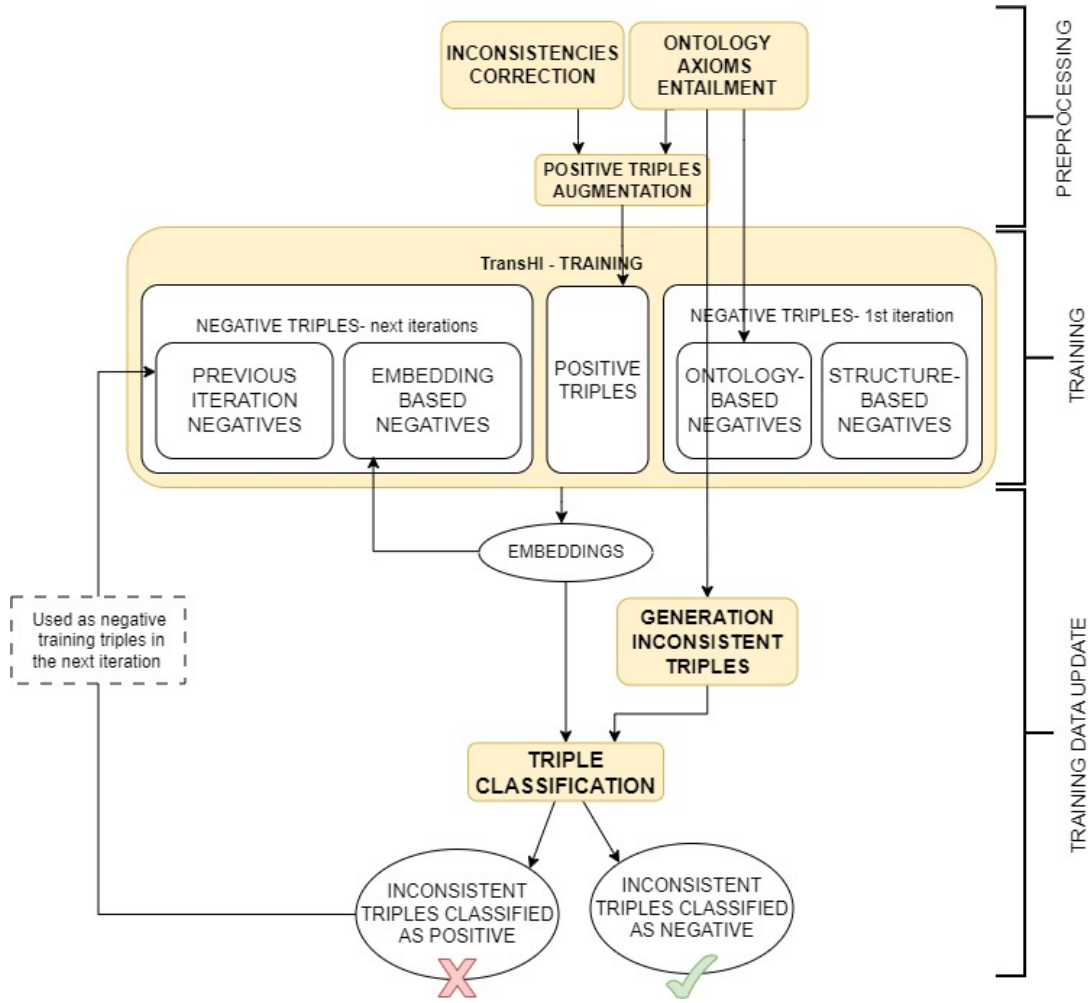


Figure 4.1: Schema of the TransHI approach.

In summary, TransHI is a method that directly addresses the challenges inherent in state-of-the-art algorithms and will be tested in Chapter 5 through its application to three real-world KGs characterized by very diverse features.

## 4.2. TransHI - A general overview

In this section, a schematic and general overview of the entire pipeline proposed by TransHI, which consists of several steps, is provided. In Fig. 4.1, the diagram of the TransHI's pipeline is represented.

Specifically, it comprises three distinct stages as depicted in the schematic:

- Preprocessing phase (further detailed in Section 4.3): An initial step conducted only once to prepare the KG for the training phase. It incorporates the following steps:
  - Inconsistencies Correction (Subsection 4.3.1): The KG is refined using its respective

ontology to identify inconsistencies, and an empirical heuristic is applied for their removal.

- Ontology Axioms Entailment (Subsection 4.3.2): The ontology is analyzed, and various types of axioms, including the entailed ones, are extracted. These will be utilized during the other steps of the pipeline.
- "Positive Triples Augmentation" (Subsection 4.3.3): Triples in the training dataset are augmented using axioms extracted from the ontology for inference purposes.
- Training phase: Employing a novel training algorithm, the KG embeddings are learned. As visible in the diagram, it leverages positive triples, including those added in the augmentation step, and negative triples. The procedure for generating effective negative triples varies between the first iteration, elaborated in Section 4.4, and subsequent iterations, detailed in Section 4.6:
  - First iteration negatives (Subsections 4.4.1 and 4.4.2): They are formulated using both axioms contained in and inferred from an ontology and by leveraging the structural properties of the graph.
  - Negatives for subsequent iterations (Subsection 4.6): Some of these are triples crafted in the preceding iteration. Others are created using an experimental heuristic, exploiting the embeddings created in the prior iteration.

The embeddings in the first iteration are initialized randomly and in the subsequent ones, with the ones learned in the previous iteration.

- Training Data Update phase (further detailed in Section 4.5): This final stage is essential for producing new negative triples to use in subsequent trainings. In particular, a set of triples inconsistent with the KG is generated using the training set and the ontology. These inconsistent triples are then used as test data in a task of triple classification. This task uses the embeddings previously generated to classify the inconsistent triples as positive or negative. Given that these triples are inconsistent, in an ideal scenario they should be all classified as negative. Every triple classified as positive is therefore misclassified. This implies that during the learning process, the entities and/or relationships contained in such triples were not accurately learned. By employing these misclassified triples as negative triples in a subsequent training, one effectively "corrects" the related embeddings that were not adequately learned.

As visible in the diagram, the step following the update of the training data is another training session, followed by another update of the training triples, and so forth, in an iterative fashion.

The iterative process stops under two conditions:

1. The generable inconsistent triples, intended to be used as test data for classification, are exhausted.

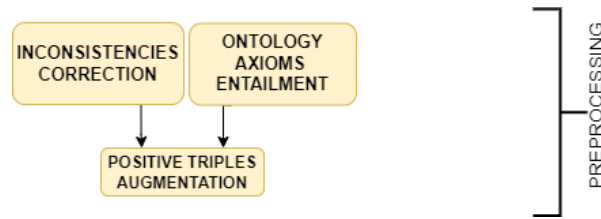


Figure 4.2: Schema of the preprocessing phase of TransHI.

2. The classification algorithm correctly categorizes all the inconsistent triples. In this scenario, it would be pointless to proceed with subsequent training since there are no specific embeddings to "correct".

Upon the pipeline's completion, one can select the best embeddings obtained across the various iterations and deploy them in the multiple embeddings real-world applications discussed in Section 2.5.1.

Subsequent sections will delve deeper into every step of the pipeline. For experimental findings, refer to Chapter 5.

### 4.3. TransHI - Preprocessing

The preprocessing phase of TransHI (represented in Figure 4.2) is dedicated to the preparation of the KG for training. This phase is crucial to ensure the reliability and trustworthiness of the KG and enhancing the effectiveness of the subsequent steps in the pipeline. Specifically, the "Inconsistencies Correction" step leverages ontologies to eliminate incorrect triples present in the database. To compensate for the removal of triples, it was considered to conduct a "Positive Triples Augmentation" phase, using the axioms of the ontology. As will be detailed further shortly, this step is particularly pivotal because, having removed inconsistencies from the KG prior to its execution, the newly produced triples will not only be accurate but also coherent, thereby preventing the addition of erroneous or misleading triples. The axioms employed in this phase encompass not only those explicitly present in the ontology but also those that can be inferred from them. Prior to the augmentation phase, there is an extraction process of such axioms from the ontology. These extracted axioms are used not only in the augmentation phase but also subsequently in the pipeline. Detailed descriptions of these three steps are provided below.

#### 4.3.1. Inconsistencies Correction

In Chapter 2.3.4, the concept of graph inconsistency when combined with an ontology was presented. Real-world Knowledge Graphs (KGs) can exhibit various inconsistencies stemming from factors such as inaccurate manual insertion, automated KG creation from the web (which

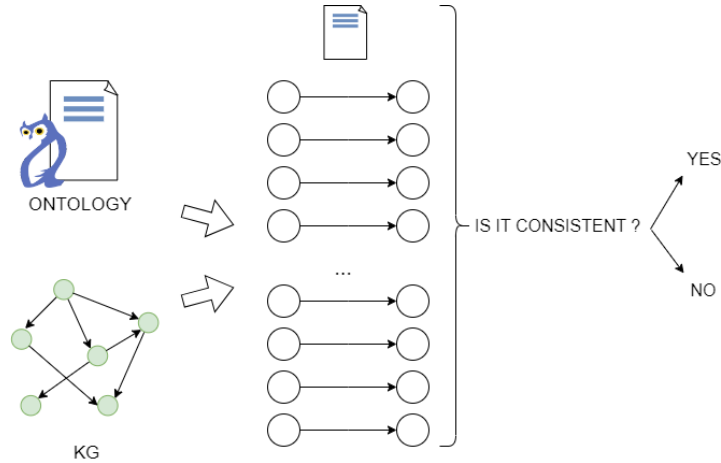


Figure 4.3: Consistency check representation.

may contain conflicting information), or the integration of multiple databases. The objective of this step is to identify KG triples that give rise to inconsistencies and effectively remove them, while minimizing the loss of information. Thus, the approach revolves around achieving essential yet minimal removals to obtain a KG that is consistent with the respective ontology.

Key aspects of this step are examined: the identification of inconsistent triples through a binary search for consistent subgroups and their subsequent removal.

- **Binary Search for Consistent Subsets** To identify inconsistent triples, a reasoning mechanism can be employed. The chosen reasoner for the practical implementation of the framework is HermiT, selected due to its capability to reason over highly complex ontologies [26]. Specifically, this reasoner is capable of determining whether a set of axioms is consistent or not. A set of axioms is considered consistent when no individual axiom or ensemble of axioms gives rise to inconsistencies. During the reasoning process, HermiT takes into account not only the axioms present in the ontology but also any that can be entailed from them.

To ascertain whether a group of triples is consistent with an ontology, the triples can be transformed into axioms of the "Assertion" type (as seen in Table 2.1 and Section 2.3.2), and it can be checked if the set composed of these axioms along with the ontology's axioms is consistent or not. To verify if an entire graph is consistent with an ontology, it suffices to check if the triples comprising it, when combined with the ontology's axioms, form a consistent set (see Figure 4.3<sup>1</sup>). It is important to note that this reasoning holds only if the ontology itself does not contain inconsistent axioms; otherwise, the combination of ontology and other axioms will always be inconsistent.

<sup>1</sup>Image created with OWL symbol from "https://perso.liris.cnrs.fr/pierre-antoine.champin/2019/ontologies/"



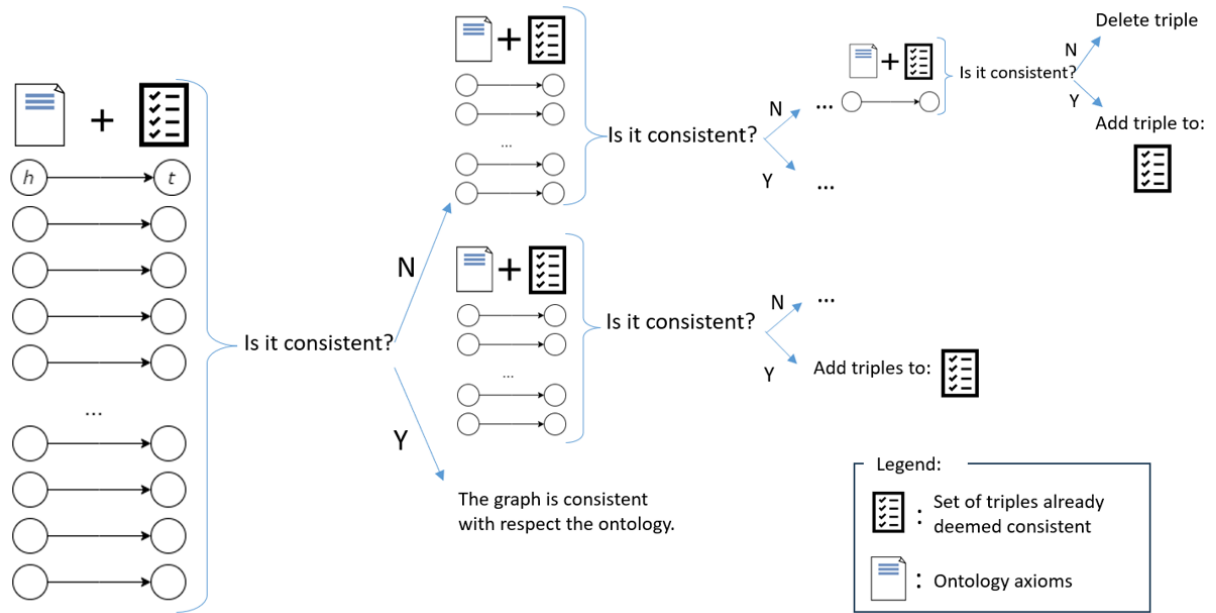


Figure 4.4: Binary search for consistent subgroups representation.

If, after conducting this check, the graph is deemed consistent, this step can be considered complete. Otherwise, it becomes necessary to identify the triples in the graph that give rise to the inconsistencies. It should be noted that the assessment must encompass not only whether each triple is consistent with the ontology's axioms but also whether all triples are consistent with each other.

An efficient method devised for this verification process is the **Binary Search for Consistent Subsets**, depicted in Figure 4.4.

A simplified version of the pseudocode for this method is presented in Alg. 4.1. The search starts by examining the consistency of all triples in the knowledge graph with the ontology (function "*isConsistent*" in Alg. 4.1). If these triples are consistent, then the entire graph is consistent with respect to the ontology (line 2 in Alg. 4.1). If not, the set of triples is divided in half (lines 7,8), and the consistency of each half, combined with the ontology and the triples identified as consistent thus far, is checked. This process is then repeated for each of the two halves (lines 9,10). If a half is found to be consistent with the ontology and the triples found consistent thus far, the triples within it are saved as consistent; otherwise, the set of triples under consideration is divided again into two, and the process continues until consistent subsets are obtained, or individual inconsistent triples are reached (line 12). Obviously, the set of consistent triples will remain empty until a subset consistent with the ontology is found. This procedure ensures that, upon completion, the found consistent triples are consistent with the ontology and with each other.

---

**Algorithm 4.1** Binary search for consistent subgroups.

---

**def** `binarySearch`(*triplesToCheck*, *ontology*, *consistentTriples*):

```

1: if isConsistent(triplesToCheck, ontology) then
2:   consistentTriples += triplesToCheck
3: else
4:   size ← size(triplesToCheck)
5:   if size > 1 then
6:     middle ← length_of_candidates ÷ 2
7:     leftHalf ← triples from triplesToCheck[0] to triplesToCheck[middle - 1]
8:     rightHalf ← triples from triplesToCheck[middle] to triplesToCheck[size - 1]
9:     binarySearch(leftHalf, ontology, consistentTriples)
10:    binarySearch(rightHalf, ontology, consistentTriples)
11:   else
12:     The triple is inconsistent.
13:   end if
14: end if

```

---

In the worst-case scenario, where all triples ( $n$ ) are inconsistent, the number of checks performed is approximately:

$$1 + 2 + 4 + \dots + \frac{n}{4} + \frac{n}{2} + n \approx 2n$$

During the binary search the sets checked to assess their consistency are smaller and smaller starting from a check on the whole set of triples in the KG ( $n$  triples). Considering that the consistency check of the whole set of  $n$  triples has a complexity of  $f(n)$ , it is reasonable to assume that checking fewer than  $n$  triples is less computationally expensive than checking  $n$ . Thus, the complexity of a binary search in the worst case is:

$$O(2n \times f(n)) = O(nf(n))$$

This approach varies linearly with the number of triples and is significantly more efficient than checking the consistency of all possible subsets of the initial set of triples (which would be  $2^n$ , an exponential complexity in  $n$ ).

#### • Removal of Inconsistent Triples

The mechanism of binary search for consistent subsets presents a fundamental issue. Assume that within a set of triples, there are only two triples that create an inconsistency when considered together. Consider the case where one of these triples is in the first half and the other in the second half of the inspected set of triples. The entire set will be detected as inconsistent. Then, when it is divided into two halves, the first inspected half will be consistent because the individual triples are not inconsistent with respect to the ontology: they create inconsistency when considered together with the other half of

triples. The second inspected half, on the other hand, will be evaluated relative to the ontology combined with the set of triples already detected as consistent, which will be containing the first half. As a result, this second half will be detected as inconsistent and will be iteratively divided to isolate the inconsistent triple, which will be discarded. This becomes a problem when this last triple is the one that is more logically correct, based on common knowledge, among the two that cause the inconsistency. Thus, one would prefer to discard the other triple.

A practical example might help illustrate this scenario. Representing each triple ( $h \in V$ ,  $r \in E$ ,  $t \in V$ ) of a KG  $G=(V,E)$  using the syntax  $(h, r, t)$ , consider that in a set of triples under examination, only these two triples cause an inconsistency:

- A)  $(TomCruise, type, Island)$
- B)  $(TomCruise, spouse, NicoleKidman)$

Now, assume that the following fragments are present in the considered ontology:

```
<! -- http://.../spouse -- >
< owl:ObjectProperty rdf:about = "http://.../spouse" >
< rdfs:subPropertyOf rdfs:resource = "http://../sameSettingAs" / >
< rdfs:domain rdfs:resource = "http://.../Person" / >
< rdfs:rangerdf:resource = "http://.../Person" / >
< rdfs:comment xml:lang = "en" > the person they are married to < /rdfs:comment >

...
< /owl:ObjectProperty >
<! -- http://.../Island -- >
< owl:Class rdf:about = "http://.../Island" >
< rdfs:subClassOf rdfs:resource = "http://.../PopulatedPlace" / >
< owl:disjointWith rdfs:resource = "http://.../Person" / >
...
< /owl:Class >
```

As can be observed, the ontology states that:

- Classes "*Island*" and "*Person*" are disjoint (Axiom\_1).
- The domain of relation "*spouse*" is "*Person*" (Axiom\_2).

The reasoner, therefore, when provided with the ontology axioms and the two triples, will make the following inferences:

- B, Axiom\_2  $\models (TomCruise, type, Person)$

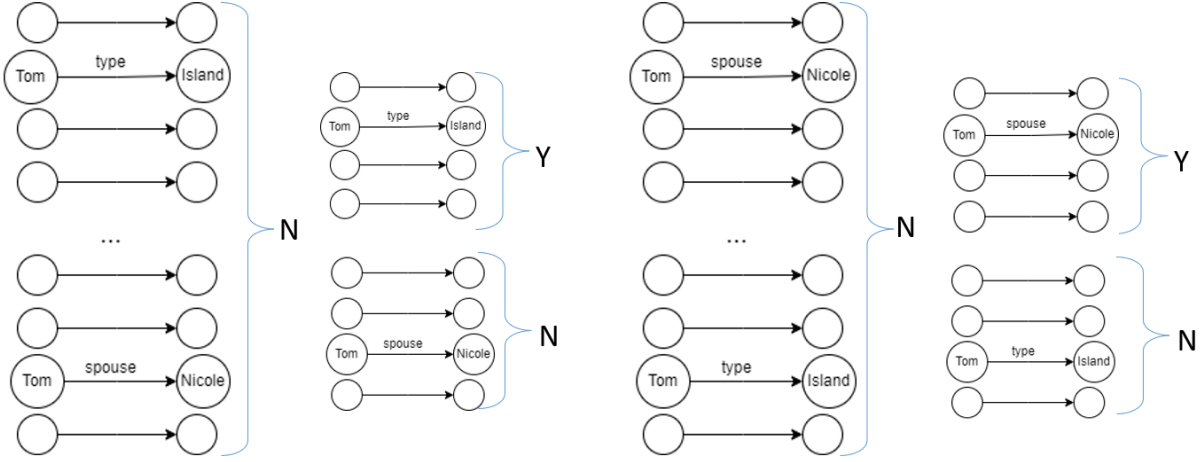


Figure 4.5: Example of the problematic related to the binary search method.

- A, Axiom\_1, (*TomCruise*, *type*, *Person*)  $\models$  Inconsistency

The set of the two triples with the ontology will thus become inconsistent. However, when the two triples are separated into two different subgroups, the first one inspected (either A or B, depending on the order of inspection) will be considered consistent, and the second one will not (illustrated in Figure 4.5).

The problem arises when the ontology lacks information about the individuals present in the triples in question but consists only of T-Box and therefore does not know whether the entity "*TomCruise*" should be considered an instance of class "*Island*" or "*Person*".

The choice of an entity's class is crucial for deciding which other types of triples are accepted/rejected. Returning to the example, if the entity "*TomCruise*" was recognized as belonging to the class "*Island*", all triples involving that entity and requiring it to be of class "*Person*" to be true would be eliminated. Therefore, it is essential to find a procedure to automatically choose the most "correct" class for each entity. To achieve this, an experimental method based on the assumption that in every KG, there are more correct triples than incorrect ones is proposed. Specifically, the hypothesis is that there are more triples that testify to an entity's membership in the correct class than the wrong class. Within the context of this new framework, the term "**evidence**" is introduced to refer to the 'proof' that can be extracted from the triplets of a KG, attesting the membership of an entity to a specific class. In particular, given an entity 'e', there is "evidence" that it belongs to a class 'C' when in the studied KG  $G=(V,E)$ , there exists  $(h \in V, r \in E, t \in V)$  such that:

- $r = \text{"type"} \text{ AND } (t = C \text{ OR } t\text{-subClass} \rightarrow C \text{ OR } x\text{-Equiv.Class} \rightarrow C)$
- $r\text{-Domain} \rightarrow x \text{ AND } (x = C \text{ OR } x\text{-subClass} \rightarrow C \text{ OR } x\text{-Equiv.Class} \rightarrow C)$
- $r\text{-Range} \rightarrow x \text{ AND } (x = C \text{ OR } x\text{-subClass} \rightarrow C \text{ OR } x\text{-Equiv.Class} \rightarrow C)$

Thus, the ontology is used to extract valuable class-related information from triples, not just those with relation "type".

The practical procedure to determine the most probable class for each entity is presented in the pseudocode 4.2 and can be summarized in three main steps:

1. For each entity, a list of counters is created, each initialized to zero, corresponding to the number of present classes (line 2 in Alg. 4.2). These counters are designated to monitor the amount of 'evidence' for the entity's membership in each specific class.
2. All triples in the KG are scanned once (lines 5-14). Whenever there is 'evidence' suggesting that an entity belongs to a class, the pertinent counter is incremented.
3. Upon completion of the scan, for each entity, the class associated with its highest value counter indicates its most probable membership class based on the 'evidence' within the KG (lines 17-22).

A simple graphical illustration of this process can be found in Figure 4.6, where the counters for the entity "*Amsterdam*" are observed to increase as the KG triples are scanned. In this representation, the triples are indicated with numbers from 1 to 5 and beneath each one, there are corresponding axioms that suggest the presence of "evidence" in that triple. The first triple, for example, contains "evidence" attesting to its membership in the class "*Agent*". Furthermore, the ontological axiom provides the information that the latter is equivalent to the class "*Q215627*". Thus, on their left, the counters for these two classes can be observed to have increased by '1'.

Utilizing the delineated procedure, one can determine the most probable and overarching class for a given entity. Indeed, when evidence is encountered for a class  $C$ , such evidence originates from a triple containing  $C$ , an equivalent class, or a subclass of  $C$ . Consequently, each encounter with a subclass of  $C$  also incrementally enhances the counter for  $C$ . As a result,  $C$  will ultimately manifest a score equivalent to or greater than that of its subclasses.

Integrating this process with binary search simply requires it to be conducted before the binary search. After determining the classes for each entity, they can be converted into assertion axioms and included in the set of triples already detected as consistent. Thus, from the initial consistency check, the identified triples remain

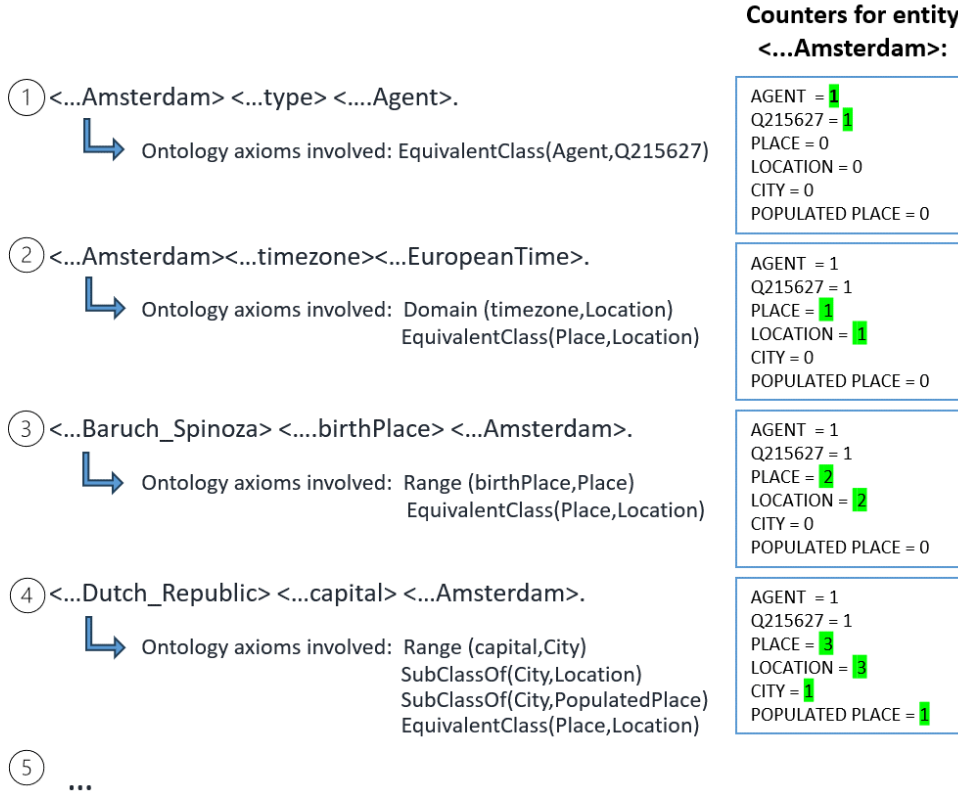


Figure 4.6: Example of the method to collect the evidence supporting class membership.

consistent with these axioms. For instance, before starting the binary search, the assertion denoting the most probable class of "*TomCruise*" should be added to the set of triples already detected as consistent:

– (*TomCruise*, *type*, *Person*)

Although these axioms are marked consistent, they are not incorporated into the KG. After cleaning the KG, they will not be listed among the consistent triples. Their role is merely to serve as "pointers" to highlight which axioms are more appropriate for removal.

Through this method, there's not only the advantage of preserving triples that are logically more accurate based on the evidence in the graph, but also an increased number of saved triples compared to not applying this technique. Intuitively, class assignments lean towards the class testified by the majority of triples. As a result, more triples will align with this class assignment than any other.

This correction stage is pivotal in ensuring that the following steps rely on accurate data.

After obtaining a dataset without inconsistencies, it can be partitioned into training and test sets. Segmenting the dataset into these subsets is a typical approach in Machine Learning. This

---

**Algorithm 4.2** Complete procedure for the removal of inconsistent triples

---

**def RemoveInconsistent**(*KG, ontology*):

```

1: Initialize consistentTriples[]
2: Initialize counters[][]
3: allClasses= list of all the classes contained in KG
4: allEntities= list of all the entities contained in KG
5: for (h,r,t) in KG do
6:   for class in allClasses do
7:     if presenceEvidence(h,class) then
8:       counter[h][class]++
9:     end if
10:   if presenceEvidence(t,class) then
11:     counter[t][class]++
12:   end if
13: end for
14: end for
15: for entity in allEntities do
16:   mostProbableClass=0
17:   for class in allClasses do
18:     if counter[t][mostProbableClass]<counter[t][class] then
19:       mostProbableClass=class
20:     end if
21:   end for Add triple (entity,type,mostProbableClass) to consistentTriples
22: end for
23: binarySearch(triplesToCheck, ontology, consistentTriples)

```

---

distinction facilitates training a model (in this context, embeddings) using only a portion of the data, with the remaining portion used to assess the model's quality. This partitioning is crucial since the quality of models must be assessed on a dataset that has not been previously encountered. Evaluating the model based on the data used for its creation would render the performance assessment biased, offering no guarantee of the model's capability to generalize accurately on novel, unseen data. In the context of embedding learning, this implies that embeddings should be evaluated on triples not utilized during the training phase. Failing to do so would leave no assurance that the embeddings of entities and relations are genuinely representative of the entities they denote, rather than being mere results of overfitting to the specific training triples. Ensuring that embeddings accurately depict the semantic relationships among entities in a knowledge graph, and their ability to generalize beyond isolated training set instances, is a quintessential characteristic sought in embeddings.

From a pragmatic standpoint, this partition typically entails the use of 70-80% of the triples for training and the remaining 20-30% for testing. It is also important that this division is random: both sets should maintain a similar distribution to preclude any potential bias in both model training and evaluation. Upon executing this partition, one can proceed to the subsequent

phases of the framework.

### 4.3.2. Ontology Axioms Entailment

If the original KG is inconsistent with its ontology, the consistent triples are fewer than the initial ones. To address this reduction, the idea is to introduce a "positive triple augmentation" step, to increase the number of triples in the training set. To perform this augmentation, the ontology axioms will be leveraged once again.

In the outlined approach, the HermiT reasoner might be leveraged for this specific step. This would involve proposing every possible entity pair connected by each potential relationship from the KG and querying the reasoner about the inferability of the triples formed from such combinations, given the KG and the ontology. However, this operation might be exceedingly time-consuming and computationally demanding: the number of inferable combinations might be minimal compared to all possible ones, rendering this approach not ideal.

Thus, the decision was made to export the ontological axioms and to use them for generating new triples by conducting "active" inferences: only inferable triples are produced with the inferences detailed in section 4.3.3, eliminating unnecessary checks. To exploit the ontology's semantic depth, not only were explicit axioms extracted, but selected inference processes were also applied to infer new ones.

This axiom exportation process was not only conducted for the subsequent "Positive Triples Augmentation" step. Two subsequent pipeline phases, training and "Generation of Inconsistent Triples", also utilize these axioms. For both, it was more time-efficient to work with exported axioms from within an ontology, mainly due to environment incompatibilities; given that HermiT is only compatible with Java and these phases employ C++ for computational speed, bridging the two languages proved to be very time-consuming. Exporting the axioms, for this additional reason, represented a preferable choice.

The subsequent analysis will focus on the types of axioms extracted and the inferences conducted.

For ease of understanding, the notation used in Tables 2.3, 2.1 and 2.2 is employed to express the axioms. For instance, the notation  $a - Domain \rightarrow x$  is used to indicate that class  $x$  is in the domain of property  $a$ , while  $a - type \rightarrow Reflex$  indicates that property  $a$  has the characteristic of being reflexive. Additionally,  $x, y, z$  represent classes, and  $a, b, c$  represent properties.

In particular, the extracted axioms are:

- Axioms about classes: Subclasses, Equivalent classes, Disjoint classes;
- Axioms about properties: Sub-properties, Domains of properties, Ranges of properties, Equivalent properties, Inverse properties, Disjoint properties, Transitive properties, Sym-



metric properties, Asymmetric properties, Reflexive properties, Irreflexive properties, Functional properties, Inverse functional properties;

As for their entailments performed, the inferences in Appendix A are applied, which are valid for standard OWL semantics, to extract entailed axioms from an ontology's axioms. Only a select few are referenced here for the sake of clarity and comprehension.

$$x\text{-Equiv.Class} \rightarrow y \models y\text{-Equiv.Class} \rightarrow x \quad (4.1)$$

$$x\text{-Disj.Class} \rightarrow y \models y\text{-Disj.Class} \rightarrow x \quad (4.2)$$

$$x\text{-Equiv.Class} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-Equiv.Class} \rightarrow z \quad (4.3)$$

$$x\text{-SubClass} \rightarrow y, y\text{-SubClass} \rightarrow z \models x\text{-SubClass} \rightarrow z \quad (4.4)$$

$$x\text{-SubClass} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-SubClass} \rightarrow z \quad (4.5)$$

$$x\text{-Disj.Class} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-Disj.Class} \rightarrow z \quad (4.6)$$

$$x\text{-Disj.Class} \rightarrow y, z\text{-SubClass} \rightarrow x \models z\text{-Disj.Class} \rightarrow y \quad (4.7)$$

$$a\text{-Domain} \rightarrow x, x\text{-Equiv.Class} \rightarrow y \models a\text{-Domain} \rightarrow y \quad (4.8)$$

$$a\text{-Domain} \rightarrow x, x\text{-SubClass} \rightarrow y \models a\text{-Domain} \rightarrow y \quad (4.9)$$

$$a\text{-Range} \rightarrow x, x\text{-Equiv.Class} \rightarrow y \models a\text{-Range} \rightarrow y \quad (4.10)$$

$$a\text{-Range} \rightarrow x, x\text{-SubClass} \rightarrow y \models a\text{-Range} \rightarrow y \quad (4.11)$$

$$a\text{-type} \rightarrow \text{Inv.Funct.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Inv.Funct.} \quad (4.12)$$

Practical examples of these entailments can be found in Figure 4.7. In particular, in the example 4.7a, to obtain the entailed axioms 4.8, 4.9 and 4.10 are used. In the example 4.7b, instead, 4.2, 4.5, 4.6 and 4.7 are used. In Fig.4.7a, the axioms already present were "*Park* – *SubClass* → *Place*" and "*Location* – *Equiv.Class* → *Place*". Using the inference A.10 the additional axiom "*Park* – *SubClass* → *Location*" can be inferred.

The example in Fig. 4.7a might not be immediately clear. One might be inclined to think that instead of adding the superclass of "*Person*" to the domain, a subclass should be added. This reasoning is conceptually incorrect: if a class belongs to the domain of a relationship, it indicates that the head of all triples with that relation belong to that class. By adding the subclasses of "*Person*" to the domain of "*ActedIn*", such as "*Female*", it would imply that the head of every triple with the "*ActedIn*" relation belongs to "*Female*", which is patently incorrect-consider, for instance, the triple (*TomCruise*, *ActedIn*, *Mission : Impossible*). However, it is accurate to say that the heads of such triples all belong to the superclasses of "*Person*", e.g. "*Mammal*".

It is important to note that the inference procedure is carried out exhaustively: instead of applying the inferences only once to the original ontology axioms, the inference rules are applied iteratively. In other words, whenever a new axiom is produced, it is used, along with others, to derive new inferences (if possible). This process continues until no new axioms can be inferred.

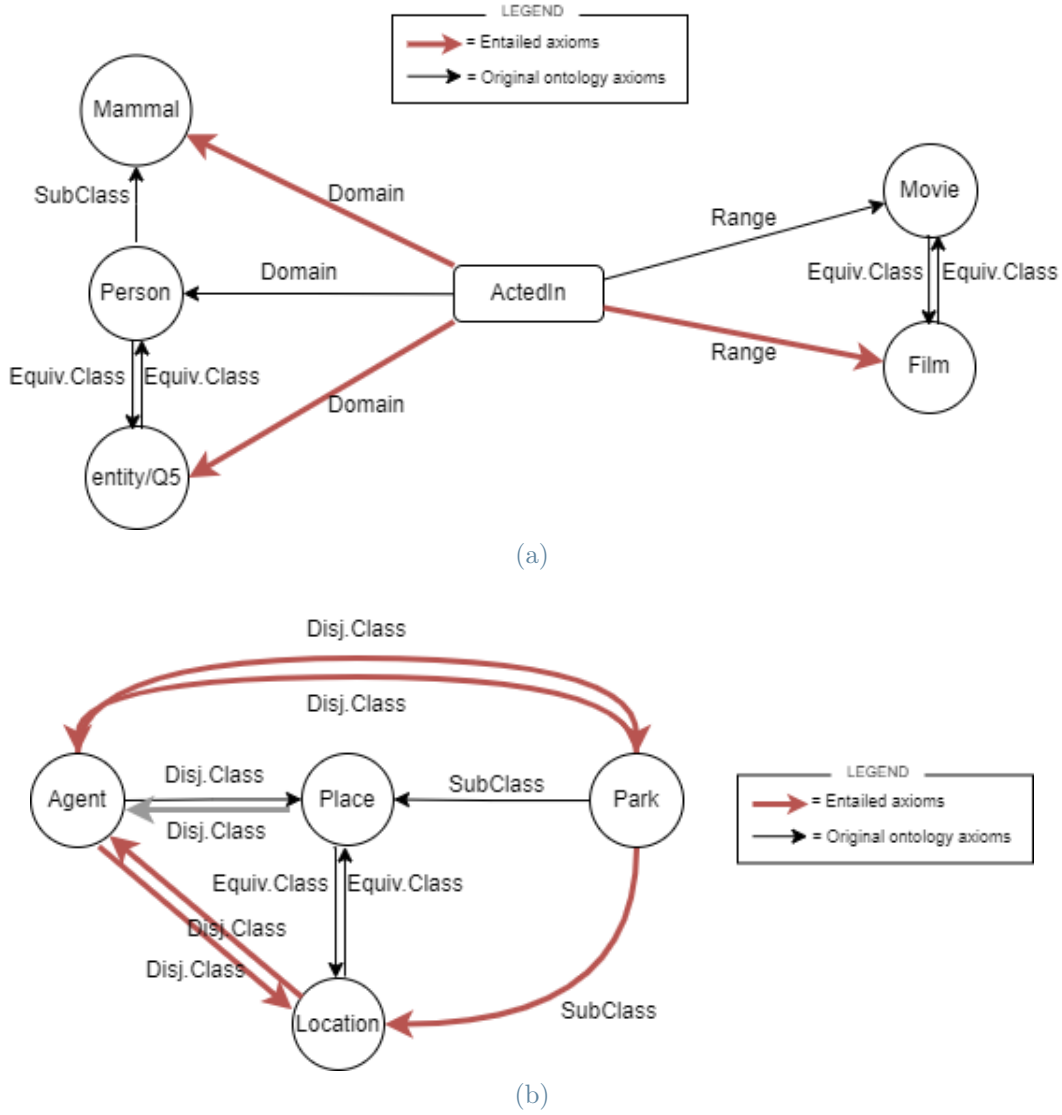


Figure 4.7: Examples of ontological axioms entailments.

To further clarify, considering the example in Figure 4.7b, after inferring the axiom "*Park* – *SubClass* → *Location*", in the next iteration, if there was, for instance, an axiom "*ThemePark* – *SubClass* → *Park*", this axiom could be used to infer "*ThemePark* – *SubClass* → *Location*".

As previously mentioned, these extracted axioms will be used both in the "Triple Augmentation" phase presented in the next paragraph and in the subsequent phases. The extensive utilization of these axioms illustrates how ontologies are extensively exploited within the newly presented approach.

### 4.3.3. Positive Triples Augmentation

The first step of the pipeline that uses the axioms extracted from the ontology is the "Positive Triples Augmentation" step. In this phase, some of the extracted ontological axioms are used to generate new positive triples. By employing new inference procedures that combine these axioms with KG triples (previously corrected in step 4.3.1), logical and consistent assertions can be obtained with respect to the ontology. It is important to recall that the process of inference can be performed not only using axioms concerning the TBox but also assertive axioms to create new information about individual instances.

As previously mentioned in the preceding section, this step is undertaken utilizing the axioms extracted from the ontology.

In machine learning algorithms such as those used for learning KG embeddings, having numerous and high-quality training data is essential for building reliable models. More training data often aids the learning: having a large, diverse and representative dataset reduces the chance of overfitting and improves generalization on test and unseen data. The quantity of data is not the sole important factor: incorrect or non-representative data can hinder the accurate model learning, even with a large training dataset.

In the context of transHI, enriching the training data using ontologies means integrating semantic patterns from ontologies into the KG data. This allows the learned embeddings to not only capture facts between instances but also to embed information on the structure of the domain of knowledge, learning regularities and patterns that might not be captured using only the original data. In summary, this process increases the number of training data as well as their semantic depth.

It is important to emphasize that the new triples are consistent with the ontology and do not introduce inconsistencies with existing triples. This is based on the use of:

- A consistent ontology - The consistency of the ontology, i.e., the absence of contradictory axioms, is crucial and is already a prerequisite for the step of "Inconsistencies Correction"(4.3.1).
- Consistent KG triples - The step of "Inconsistencies Correction"(4.3.1) is specifically designed to ensure that the KG is consistent with the ontology.

Starting from valid triples and axioms, the newly inferred triples will inherit their validity and coherence. This is a fundamental principle of deductive logic: starting with true premises and applying correct reasoning leads to true conclusions. In this context, the true premises are the triples of the KG and axioms of the ontology, the reasoning comprises inference rules, and the conclusions are the new triples.

Denoting  $e1, e2, e3$  as entities in the analyzed KG,  $x, y, z$  as classes, and  $a, b, c$  as properties, the following inferences are conducted to obtain the new triples:

$$e1\text{-type} \rightarrow x, x\text{-Equiv.Class} \rightarrow y \models e1\text{-type} \rightarrow y \quad (4.13)$$

$$e1\text{-type} \rightarrow x, x\text{-SubClass} \rightarrow y \models e1\text{-type} \rightarrow y \quad (4.14)$$

$$e1\text{-a} \rightarrow e2, a\text{-SubProp} \rightarrow b \models e1\text{-b} \rightarrow e2 \quad (4.15)$$

$$e1\text{-a} \rightarrow e2, a\text{-Equiv.Prop} \rightarrow b \models e1\text{-b} \rightarrow e2 \quad (4.16)$$

$$e1\text{-a} \rightarrow e2, a\text{-Inverse.Prop} \rightarrow b \models e2\text{-b} \rightarrow e1 \quad (4.17)$$

$$e1\text{-a} \rightarrow e2, a\text{-type} \rightarrow \text{Symm.} \models e2\text{-a} \rightarrow e1 \quad (4.18)$$

$$e1\text{-a} \rightarrow e2, e2\text{-a} \rightarrow e3, a\text{-type} \rightarrow \text{Trans.} \models e1\text{-a} \rightarrow e3 \quad (4.19)$$

$$e1\text{-a} \rightarrow e2, e1\text{-a} \rightarrow e3, a\text{-type} \rightarrow \text{Funct.} \models e2\text{-SameAS} \rightarrow e3 \quad (4.20)$$

$$e1\text{-a} \rightarrow e2, e3\text{-a} \rightarrow e2, a\text{-type} \rightarrow \text{Inv.Funct.} \models e1\text{-SameAS} \rightarrow e3 \quad (4.21)$$

Practical examples of the triples that can be obtained by applying these inferences include:

- (Using inference 4.16)  $(\text{Karlsruhe}, \text{District}, \text{Germany}), \text{District-SubPropOf} \rightarrow \text{isPartOf} \models (\text{Karlsruhe}, \text{isPartOf}, \text{Germany})$
- (Using inference 4.17)  $(\text{LionelRichie}, \text{HasChild}, \text{NicoleRichie}), \text{HasChild-Equiv.Prop} \rightarrow \text{HasOffspring} \models (\text{LionelRichie}, \text{HasOffspring}, \text{NicoleRichie})$
- (Using inference 4.14)  $(\text{SanMarino}, \text{type}, \text{Location}), \text{Location-Equiv.Class} \rightarrow \text{Place} \models (\text{SanMarino}, \text{type}, \text{Place})$
- (Using inference 4.15)  $(\text{SelenaGomez}, \text{type}, \text{Actor}), \text{Actor-SubClassOf} \rightarrow \text{Person} \models (\text{SelenaGomez}, \text{type}, \text{Person})$

Once these inferences are applied to the consistent KG triples, the inferred triples are added to the training dataset if they were not part of it already.

One point to highlight is that performing the inconsistency removal step before this is crucial. If only the augmentation step were performed, new false triples could have been inferred, further corrupting the database.

#### 4.4. Trans-HI - Training in first iteration

Once the database has been corrected in the "Inconsistencies Correction" phase and the number and semantic depth of training triples has been increased in the "Positive Triples Augmentation" step, the training dataset is ready for being used to learn embeddings.

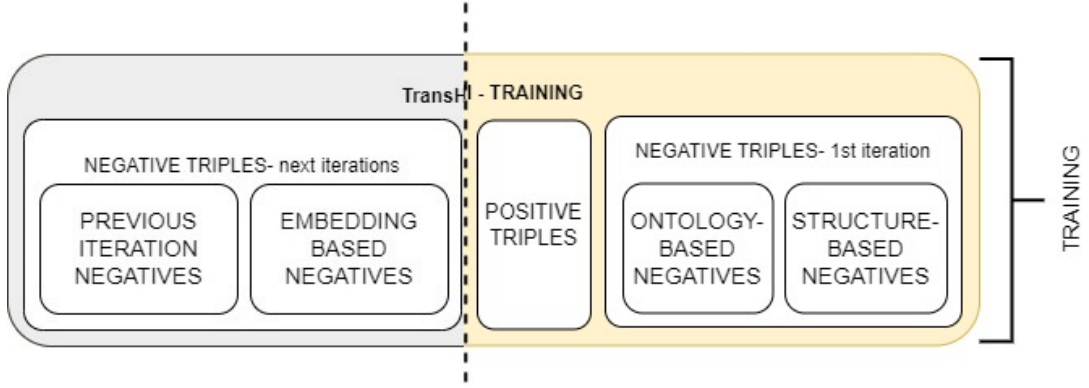


Figure 4.8: Schema of the TransHI component for the first iteration training.

The training algorithm used in TransHI is schematized in Figure 4.8, that highlights the part related to the first iteration of the training. The training algorithm is built upon the foundation of TransE. The innovation in this model lies in how negative triples are created and the approach used is hybrid and iterative. In the first iteration, two distinct methodologies are employed to generate negatives: one that leverages ontological axioms and the other that utilizes the structural properties of the graph. The method of triple creation is differentiated in the first iteration compared to subsequent ones. This distinction arises because the generation of negatives in later iterations relies on the embeddings produced in previous iterations. Naturally, in the first iteration, these embeddings do not yet exist, and therefore cannot be utilized for negative creation.

This section presents the technique for generating negative triples in the first iteration of training, and section 4.6 will cover the method used in subsequent iterations.

### The Score Function

The score function, which assigns a "score" to each triple, is exactly the same as in TransE:

$$f(e_h, e_r, e_t) = \|e_h + e_r - e_t\|_2^2$$

where  $e_h$  and  $e_t$  are the head and tail entities of a triple, and  $e_r$  is the corresponding relation. Triples that are true in the knowledge graph will have a score close to 0, while false triples should correspond to higher scores. Therefore, the model aims to minimize the score of correct triples and maximize the score of negative triples. This is because, similar to TransE, the semantic relationships between entities in this model are expressed in terms of geometric distances in the embedding space.

### The Loss Function

Firstly, it should be noted that in the design of TransHI, the aim was to maintain the number of positive triples equal to the number of negative triples to achieve a model capable of effectively

discerning true from false triples. Specifically, similar to TransE, for each positive triple in the training set, a negative triple is generated.

However, the initial goal was to improve TransE: creating negative triples entirely at random constitutes a simple algorithm with room for improvement. TransE, by generating random negative triples, is prone to creating negatives that should actually be positives. These might be present in the test set or represent missing links that should exist in the graph.

A method was devised that employs the combination of two different approaches for negative triple creation:

- Ontology-based method;
- Structure-based method;

Chapter 3 already discussed how TransOWL improves TransE's performance using ontologies. However, TransOWL has a weakness that is addressed by TransHI: ontologies can provide information only about certain classes or properties in the graph. It is possible that a positive triple does not contain relations/entities such that an ontology-based negative creation is possible. Therefore, TransHI uses an hybrid approach, using a non-ontology-based method was chosen to create effective negatives for these cases.

In particular, for ontology-based negative triples, different methods are used based on whether the triple is of the "typeOf" type (associating an instance with its class using the "type" relation) or the "noTypeOf" type (involving any other relation).

These different methods for creating negative triples are reflected in the formulation of the loss function:

$$\begin{aligned}
L = & \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{Ont\_typeOf}}} \sum_{(h',r,t') \in S'_{\text{Ont\_typeOf}}} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{Ont\_noTypeOf}}} \sum_{(h',r,t') \in S'_{\text{Ont\_noTypeOf}}} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) \\
& + \sum_{(h,r,t) \in S_{\text{Structure}}} \sum_{(h',r,t') \in S'_{\text{Structure}}} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'}))
\end{aligned} \tag{4.22}$$

The four terms composing the loss function only differ in how the negative triple is generated. Specifically, in the first term, negatives are generated exactly as in TransE: corrupting either the head or the tail randomly. In the second and third terms, ontology-based axioms are exploited, and in the fourth term, the KG structure is leveraged.

The term related to random negatives is included because, as observed in many of the analyzed algorithms from TransOWL to SANS, a component of random negative triples, created as in TransE, is necessary for uniform learning, utilizing a broad range of relations and entities. This term also helps prevent the algorithm from overfitting on the structural properties or ontology axioms used. In other words, while using the two selected methods results in effective negative creation, maintaining a random component is crucial to ensure that the model does not focus excessively on specific patterns. Thus, to effectively gauge the weight of this random component, an algorithmic training hyperparameter is introduced. This parameter indicates the percentage of positive triples for which a random negative is created. The optimal percentage of randomness varies from one knowledge graph (KG) to another. To achieve optimal outcomes, it is advisable to empirically ascertain the most suitable percentage of randomness for your KG by repeatedly iterating through the initial training step while adjusting this parameter.

The loss function is minimized using gradient descent, just like in TransE. Thus, for each positive-negative triple pair, the contribution of that pair to the loss is calculated, corresponding to a single term in the summation in equation 4.22. The embeddings are updated as in TransE, using the learning rate and the partial derivatives of the loss:

$$e_{\text{new}} = e_{\text{old}} - \alpha \times \nabla_{e_{\text{old}}} L \quad (4.23)$$

In the pseudocode 4.3, the broad working principles of the training algorithm for the first iteration are outlined.

As observed in the pseudocode, an input parameter, `pct_rnd_negatives`, is utilized at line 5 to determine the number of triples constructed randomly. For those constructed using ontologies or the graph's structure, a specific percentage, `pct_OS`, dictates their ratio(line 6). The method for determining this percentage will be elucidated shortly. Moreover, triples crafted using the graph's structural properties (line 12) are also employed whenever, given a positive triple, no utilizable ontological axioms are available for negative triple construction.

In the realm of training algorithms, iterating over multiple "epochs" ('TrainTimes' in 4.3) is indispensable. An "epoch" is defined as a cycle of training through the entire training dataset. The use of multiple epochs is crucial to converge to an optimal solution that minimizes the loss function; with each epoch, the acquired models continually refine. However, it is also judicious to moderate this parameter, as an excessive number can heighten the susceptibility to overfitting.

---

**Algorithm 4.3** TransHI training function for first iteration pseudocode
 

---

**def transHI\_training\_function\_1stIter** (pct\_rnd\_negatives, pct\_OS):

```

1: Initialize embeddings randomly
2: for  $i = 1$  to TrainTimes do
3:   for  $j = 1$  to size(KG) do
4:     Sample positive triple  $(h, r, t)$  from  $S$ 
5:     if  $\text{random\_number}(0, 100) > \text{pct\_rnd\_negatives}$  then
6:       if  $\text{random\_number}(0, 100) > \text{pct\_OS}$  then
7:         Negative triple  $(h', r', t') = \text{negWithOntology}(h, r, t)$ 
8:         if  $(h', r', t')$  is not in training set and not already used then
9:            $\text{updateEmbeddings}(h, r, t, h', r', t')$ 
10:        continue
11:      end if
12:      Negative triple  $(h', r, t') = \text{negWithStructure}(h, r, t)$ 
13:      if  $(h', r', t')$  is not in training set and not already used then
14:         $\text{updateEmbeddings}(h, r, t, h', r, t')$ 
15:      continue
16:    end if
17:  end if
18: end for
19:   Negative triple  $(h', r, t') = \text{randomNegative}(h, r, t)$ 
20:    $\text{updateEmbeddings}(h, r, t, h', r, t')$ 
21: end for
22: end for
23: Output embeddings

```

---

A salient detail concerning the creation of triples via both methods is the consistent verification that the generated negative triples are not redundant. This precaution ensures that the same triples are not repeatedly used, thereby mitigating the risk of overfitting on those particular triples, enhancing the generalization capability of the embeddings on unseen data. Another crucial verification carried out on all produced negative triples, including the random ones, is ensuring they do not belong to the training set.

From an implementation perspective, to perform these two types of checks, it is imperative to have a data structure to store the positive triples of the training set and the negative triples as they are generated. Given that Knowledge Graphs (KGs) can encompass hundreds of thousands of triples, it is paramount to adopt a structure that is efficient both in terms of access time and memory usage. As such, a design was chosen that combines the benefits of adjacency lists with



those of adjacency matrices: three nested unordered maps. These maps are a data structure that offer data access speed comparable to adjacency matrices, ranging from  $O(1)$  to  $O(n)$  where  $n$  is the total number of entities. Unlike matrices, however, they do not allocate memory for non-existent edges in the KG. They only store existing edges, akin to adjacency lists. This trait, as previously discussed in Section 2.1.4, is crucial if the graph exhibits sparsity. The utilization of three nested maps allows for a more efficient segmentation of information, accelerating searches. Each of the three maps uses the subjects, predicates, and objects of the KG triples as keys. This data structure amalgamates the rapid access of adjacency matrices with the spatial efficiency of adjacency lists.

A detailed exposition of the two methods employed for negative triple construction is now presented.

#### 4.4.1. Ontology-based negative triples

Part of the triple negatives generated in the first iteration of TransHI originates from axioms extracted from the ontology (section 4.3.2). The negative triples are designed to be inconsistent with the training set and the ontology. Creating such inconsistent triples ensures their negativity. Moreover, by employing diverse ontology axioms during training, the resulting embeddings incorporate ontological patterns and can generalize these patterns to unseen data, thereby enhancing their generalization capabilities.

The axioms used pertain to the following:

- Disjoint classes
- Disjoint properties
- Irreflexive properties
- Asymmetric properties
- Domain of relations
- Range of relations

The first type of axioms serves to create "typeOf" triples, while all others are used to generate "noTypeOf" triples. In the context of this discussion, the classes of which an entity "a" is an instance in the studied KG are denoted with "Classes(a)". Conversely, "Entities(A)" encompasses entities that are instances of class A.

In Table 4.1 each negative triple type (identified by an ascending numerical ID for clarity) is analyzed individually. Starting from a positive triple (h, r, t), the corrupted element of such a triple is indicated with an apostrophe; for instance, if the head is the one being corrupted for

ID	Axiom used	typeOf/ noTypeOf positive	Corrupting h/t/r	Conditions	Corrupted entity
1	Disjointed classes	typeOf	h	$t\text{-Disj.Class} \rightarrow C$	$h' \in \text{Entities}(C)$
2	Disjointed classes	typeOf	t	$h \in \text{Entities}(C) \ \&\& \ C\text{-Disj.Class} \rightarrow D$	$t' = D$
3	Disjointed properties	noTypeOf	r	$r\text{-Disj.Prop} \rightarrow p$	$r' = p$
4	Irreflexive properties	noTypeOf	t	$r\text{-type} \rightarrow \text{Irreflexive}$	$t' = h$
5	Asymmetric properties	noTypeOf	h,t	$r\text{-type} \rightarrow \text{Asymmetric}$	$t' = h, h' = t$
6	Domain of relations	noTypeOf	h	if $r\text{-Domain} \rightarrow C$ implies $r \in \text{DomainClasses}(r)$	$\text{Classes}(h') \cap \text{DomainClasses}(r) = \emptyset$
7	Range of relations	noTypeOf	t	if $r\text{-Range} \rightarrow C$ $\implies C \in \text{RangeClasses}(r)$	$\text{Classes}(t') \cap \text{RangeClasses}(r) = \emptyset$

Table 4.1: Types of negative triples generable using ontology axioms.

the negative creation, the substituting entity is denoted by h'. The table employs uppercase letters (e.g."C", "D") to denote classes and lowercase letters(e.g."p") for properties.

It is worth noting that using the axioms of disjointness, domain, and range (in particular for negatives with ID=1,6,7) leads to the creation of many negative triples, as they are created by corrupting components of the original triple with entities belonging to certain classes, and instances of certain classes in a KG can be numerous.

Following are two practical examples of creating negative triples with ontology:

- Generation of a "typeOf" negative - Consider a positive triple "typeOf": (*TomCruise*, *type*, *Actor*). Given the extraction of axioms "*Actor* – *Disj.Class* → *Building*" and "*Actor* – *Disj.Class* → *Island*" from the KG's ontology, corruption of the triple's head (triple with ID=1) would require entities from classes disjoint with the "*Actor*" class. Consequently, instances of "*Building*" or "*Island*" would serve as head entities for negative triples.
- Generation of a "noTypeOf" negative - Consider a positive triple "noTypeOf": (*LionelRichie*, *hasChild*, *NicoleRichie*). For the creation of a negative with ID=6, all entities from classes not in the domain of "*hasChild*" can be used as heads for the negative triples. Assuming that the domain of "*hasChild*" only includes the class "*Person*", new head entities could come from classes that are not instances of "*Person*".

In the context of triples generated with domains and value ranges (ranges) it is necessary to

provide important specifications. It would be more appropriate to only use entities belonging to classes that are disjoint from those for which the domain or value range is being defined. This arises from the consideration that an entity may concurrently be an instance of several classes if they are not mutually disjoint. However, this condition would be excessively restrictive and would significantly reduce the number of generable negative triples. To overcome these limitations, an approach based on the Closed World Assumption (CWA) has been adopted. According to this assumption, all triples not present in the provided Knowledge Graph (i.e., the training set) are considered false. Therefore, if there is no triple associating an entity with a class, then that entity should not belong to that class. This assumption can be applied as, during the triple augmentation step, triples were added that associated entities with subclasses and equivalent classes of their existing classes in the KG. In particular, this assumption is applied exclusively to entities that serve as instances of specific classes. Entities that are not designated as belonging to any particular class, as will be discussed shortly, are not utilized for generating this type of negative triples.

In relation to the subclasses and superclasses associated with domains and ranges, during the axiom entailment process (as referenced in section 4.3.2), the definitions of domain and range for any given relation were expanded to encompass all superclasses and equivalent classes of those originally present in the respective domains and ranges. For example, if a relation "Painted" has a domain class of "Painter", this domain would be expanded to also include superclasses like "Person" and "Human". This is because, as elaborated in section 4.3.2, all triples associated with the relation "Painted" will have their heads belonging to classes such as "Person" and "Human". Such classes, when generating triples like ID=6, are considered as the classes from which the entity, meant to corrupt the head/tail of the positive triple, cannot be extracted to form the negative triple. This might be perceived as restrictive, given that there could exist entities of "*Human*" that are not entities of "*Painter*". However, it is also true that by strictly excluding entities not even belonging to superclasses, the likelihood of producing false negatives diminishes. This is evident when an entity representing a painter, say "Michelangelo", lacks a connection with the class "Painter" either because it is part of the test set or it is an omitted link due to incompleteness. Such an entity could be used as the head of the negative triple, inadvertently producing a false negative. By also excluding classes like "Human", this risk is mitigated.

The step of axiom entailment is fundamental as it allows the use of an extensive set of ontology axioms without requiring further complex reasoning. It is worth noting that axiom entailment has been performed for all categories of axioms mentioned in chapter 4.3.2, not just domain/range. For instance, even all disjointness axioms have been made reflexive, and subclasses and equivalent classes have been included. This is a feature that sets TransHI apart from TransOWL. TransHI operates with pre-processed ontology axioms, ensuring greater variety and accuracy in the generated negative triples.

To illustrate with a tangible example, let's consider the triple (*LionelRichie*, *HasChild*, *NicoleRichie*) and the creation of a negative triple of the type ID=5. It is known that the pertinent axioms are "*HasChild* – *Equiv.Prop* → *HasOffspring*" and "*HasChild* – *type* → *Asymmetric*". With the "Positive Triples Augmentation" step, the triple (*LionelRichie*, *HasOffspring*, *NicoleRichie*) is added. Without axiom entailment, the triple (*LionelRichie*, *HasChild*, *NicoleRichie*) would not have negatives of the type ID=5 since there is not an axiom indicating that "*HasOffspring*" is antisymmetric. In TransHI, instead the "axioms entailment" step creates the axiom "*HasChild* – *type* → *Asymmetric*" and allows during the training to create the negative (*NicoleRichie*, *HasOffspring*, *LionelRichie*).

Two details that needed to be implemented during the practical algorithm development are as follows:

- A delicate but essential point to achieve optimal performance is the need to diversify the generated negative triples as much as possible. As previously mentioned, the use of both ontologies and the KG structure is designed so that negative triples are never reused. This is a crucial step to avoid overfitting on specific entities and relations and to promote distributed learning involving a wide range of entities and relations. For negative triples generated through ontology axioms, diversification is a greater challenge. As highlighted earlier, negative triples generated using domain, value range, and disjointness axioms can rapidly multiply due to the introduction of new entities from selected classes. During the training phase, not all potential negative triples will be created, as each positive triple will be used in training together with a different corresponding negative triple an average number of times determined by the number of training epochs ("trainTimes" in the algorithm 4.3). The solution adopted is to select entities from permissible classes to extract entities from classes in a rotating manner. This means that the choice of substitutive entities to create negatives from classes should occur uniformly and be well-balanced. This strategy ensures the uniform use of classes and avoids focusing solely on specific classes. This is exemplified through an example. Taking the illustrative positive triple (*TomCruise*, *type*, *Actor*) and wanting to corrupt its head using disjointness axioms (in table 4.1, ID=1), all entities belonging to classes that are disjoint from *Actor* can be used. If "*Actor*" were disjoint from three classes, such as "*Plant*", "*Place*", and "*Settlement*" and the intention was to initially utilize all entities from the "*Plant*" class, followed by those from the "*Place*" class, and finally those from the "*Settlement*" class, there could be a potential risk that entities from the "*Place*" and "*Settlement*" classes are never utilized. This happens especially if the "*Plant*" class has a significant number of instances. In such a case, the fact that "*Actor*" is disjoint from "*Place*" and "*Settlement*" might not be learned by the embeddings; it would only be learned "*Plant*" is disjoint from "*Actor*". The solution is to extract entities from available classes (in this case, the three mentioned classes) in a way that evenly learns the properties expressed by the three disjointness

axioms.

- As noted, for triples of types ID=1, 6, 7, the corrupted entity is chosen from entities belonging to selected classes. If an entity is not an instance of any class, it is not used in generating ontology-based triples. If the percentage of entities without class is high, this can lead to low-quality embeddings for these entities as they have not been sufficiently utilized in training. The solution adopted for this problem is to calculate the percentage of entities without class and use this percentage to decide how many triples should be ontology-based and how many should be structure-based. In the pseudocode 4.3, this is precisely the percentage "pct\_OS" used in line 6. As it will be discussed shortly, structure-based triples have been designed to be used for all positive triples for which a negative cannot be found through ontology axioms. It was decided based on experimental evidence to use them also in proportion to the number of entities without classes. This is because, with the structural method, the substitute entities used to replace the head or tail of the positive are not determined by its membership in a class but rather by structural properties of the KG.

Algorithm 4.4 presents the pseudocode for the function responsible for creating negative triples using ontology axioms. The function's name is *negWithOntology*. The pseudocode contains conditional branching based on the type of relation in the positive triple (h, r, t) being considered. For instance, if the relation is 'type', the function decides between creating negatives with disjoint classes changing the head(triple with ID=1, line 3) or disjoint classes changing the tail(ID=2, line 6). If the relation is not 'type', the function prioritizes creating irreflexive properties (ID=4, line 11) and asymmetric properties (ID=5, line 14) before disjoint properties (ID=3, line 17), domain (ID=6, line 22), or range (ID=7, line 25) axioms. This order aligns with the idea that irreflexive and asymmetric properties can be used only once per positive, while the number of negatives generated using disjoint properties, domain, and range axioms can be considerably higher. The function returns a negative triple (h', r', t'). In the pseudocode, for simplicity, the check performed on the generated negative triples has been omitted: they must not be repeated throughout the entire training process.

#### 4.4.2. Structure-based negative triples

During the first iteration of TransHI, there could exist triples for which the ontology does not contain axioms that can be exploited for the creation of negative triples. This means that none of the conditions presented in Table 4.1 can be satisfied for such triples. As mentioned earlier, in such cases, along with a percentage determined by the number of entities not belonging to any class, the triples are created using a heuristic based on the KG's structure.

The structure-based method used is based on the SANS method, already discussed in Chapter 3.2. In this method, each negative triple is created by corrupting the head or tail (with a 50%

---

**Algorithm 4.4** Function pseudocode: creation of negative triples with ontology axioms

---

**def** *negWithOntology* (h,r,t):

```

1: if r=='type' then
2:   if random_number(0,100) > 50 then
3:     (h',r,t)=negWithDisjointClass_ID1(h,r,t)
4:     Return (h',r,t)
5:   else
6:     (h,r,t')=negWithDisjointClass_ID2(h,r,t)
7:     Return (h,r,t')
8:   end if
9: else
10:  if irreFlexivePropAlreadyUsed(h,r,t)==false then
11:    Return (h,r,h) # Triple type ID4
12:    irreFlexivePropAlreadyUsed(h,r,t)=true
13:  else if asymmPropAlreadyUsed(h,r,t)==false then
14:    Return (t,r,h) # Triple type ID5
15:    asymmPropAlreadyUsed(h,r,t)=true
16:  else if disjointPropertiesNotUsed(h,r,t).size()>0 && random_number(0,100)>50 then
17:    (h,r',t)=NegWithDisjointProp_ID3(h,r,t)
18:    disjointPropAlreadyUsed(h,r,t).remove(r')
19:    Return (h,r',t)
20:  else
21:    if random_number(0,100)>50 then
22:      (h',r,t)=negWithDomain_ID6(h,r,t)
23:      Return (h',r,t)
24:    else
25:      (h,r,t')=negWithRange_ID7(h,r,t)
26:      Return (h,r,t')
27:    end if
28:  end if
29: end if

```

---

probability each) of a positive triple, selecting the substitute entity from the k-hop neighbors of the entity to be replaced. A simple representation of this process, changing the tail of the positive triple (h, r, t) with a neighbor of radius 2, is depicted in Figure 4.9.

Specifically, the RW-SANS method (already presented in section 3.2 is employed due to its scalability, which is essential for processing knowledge graphs of substantial size. In the SANS method, the neighbor to replace an entity is randomly selected. The additional idea introduced by TransHI is to give meaningful order to the neighbors, and thus the potential substitutes, of a corruptible entity.

The method for ordering these neighbors was inspired by the phrase from the paper "Towards Loosely-Coupling Knowledge Graph Embeddings and Ontology-based Reasoning" [18], mentioned in Chapter 3.2: "An intuitive strategy is to explore sparse regions of the graph as entities

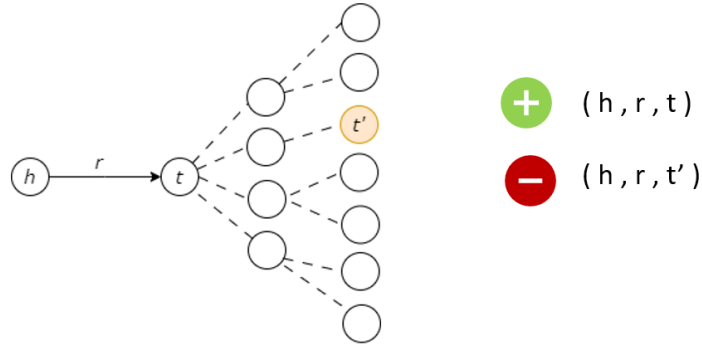


Figure 4.9: Example of application of the SANS method on a positive triple.

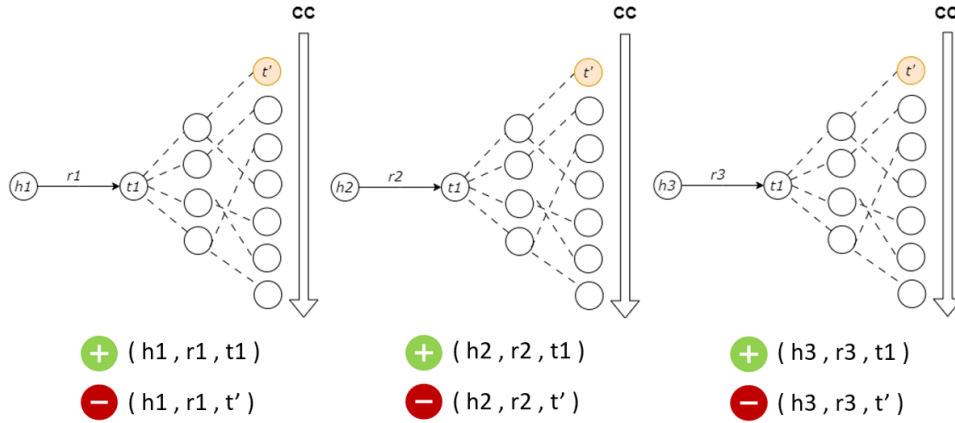


Figure 4.10: Selection of neighbors with low diversification.

that are densely connected are less probable to have missing true relations". In TransHI, neighbors are "ordered" so that those in sparser regions of the graph are given priority. To achieve this, they are simply sorted in ascending order based on their clustering coefficient (CC).

Since it is always ensured that duplicate negatives are not created, it is not possible for the head or tail to be replaced with the same neighbor more than once given a triple  $(h, r, t)$ . However, this control is not sufficient to ensure the diversification of the entities used in creating negatives. This can be understood by looking at the illustration in Figure 4.10. In the representation, it is evident that across three distinct positive triples, the entity to be corrupted is 't1'. In all these instances, it gets replaced by the entity with the lowest clustering coefficient, 't'. This becomes problematic when the entity 't1' appears in numerous positive triples, as it would be "overrepresented" during training. Simultaneously, all non-utilized neighbors, even if they have a relatively low clustering coefficient, would be underrepresented.

It is desired to achieve a more uniform distribution of entities involved in training. Therefore, the decision was made that whenever it is necessary to corrupt an entity during training, a new neighbor is considered, always following the ascending order of their clustering coefficient (CC) values. Implementing this solution would change the situation illustrated in Figure 4.10 to that

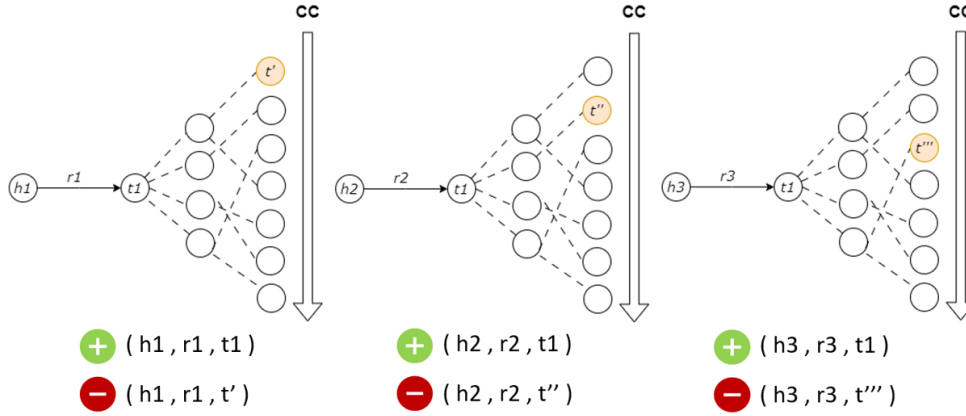


Figure 4.11: Selection of neighbors with high diversification.

shown in Figure 4.11. In these examples, it is assumed that the triples appear in chronological order during the execution of the training algorithm, as they are arranged from left to right in the images.

An important aspect to emphasize is that, just as with triples constructed using ontological axioms, the generated negative triples are never repeated. In this case, it is also ensured that they do not belong to the training set (otherwise, they would be positive). This check was not necessary for triples created leveraging the ontology, as they, being inherently inconsistent, could not be part of the training set, which was cleansed of inconsistencies at the beginning of the pipeline.

On the implementation side, for temporal efficiency, the neighbors of entities computed before the training algorithm is executed. They are then sorted based on CC values in the initial phase of the training algorithm.

The process is described in a simplified manner in the pseudocode 4.5. On lines 3 and 7, the neighbour with the lowest CC that is 'valid', meaning that it forms a triple not belonging to the training set and not already used as negative, is selected and then removed from the list of neighbors of that entity.

If the triple formed with the neighbor constitutes a triple that exists in the training set, the subsequent neighbor in order will be considered. The neighbors not used in these cases will remain eligible as substitutes for that entity in other triples. This control is not shown in the pseudo-code 4.5 for simplicity.

It should be specified that the "memorization" of which neighbors have already been used and which have not is employed throughout the training: neighbors are not "reset" from epoch to epoch. This is due to the fact that, without this mechanism, during each epoch, the first  $n$  neighbors would be repeatedly employed on average, where  $n$  represents the frequency of an entity's occurrences in the training set. Consequently, to ensure an adequate supply of neighbors



---

**Algorithm 4.5** Function pseudocode: creation of negative triples based on the structure  
**def** *negWithStructure* (h,r,t):

---

```

1: #In Neighbors[x] there are already the neighbors of entity x sorted in ascending order.
2: if random_number(0,100)>50 then
3:   (h',r,t)=First element of Neighbors[h]
4:   Remove h' from Neighbors[h]
5:   Return (h',r,t)
6: else
7:   (h,r,t')=First valid element of Neighbors[t]
8:   Remove t' from Neighbors[t]
9:   Return (h,r,t')
10: end if

```

---

across all training epochs, a substantial number of neighbors must be obtained for each entity. Employing RW-SANS as the foundation for neighbor extraction necessitates the execution of a significant number of random walks, and the neighborhood radius must be sufficiently extended to encompass a notable quantity of neighbors.

## 4.5. TransHI - Training Data Update

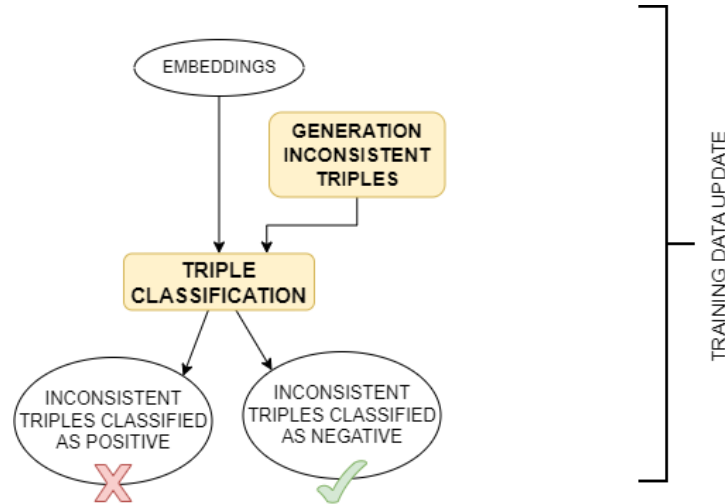


Figure 4.12: Schema of the "Training Data Update" phase of TransHI.

The output of the training process consists of embeddings for each entity and relation in the KG. In the "Training Data Update" step, schematized in Fig. 4.12, these embeddings are used in a classification task to determine whether triples can be classified as positive or negative based on their score (obtained through the generated embeddings). The main objective is to identify entities and/or relations whose embeddings were not optimally learned during the training algorithm. These can be identified by looking at triples that are misclassified.

Utilizing the training data for classification purposes would be illogical. This data has been repeatedly exposed throughout the training process, equivalent to the number of training epochs, rendering misclassification of these samples highly improbable. Naturally, test data cannot be used as the data to classify: using test data during the training process is fundamentally incorrect as it would "contaminate" the training and result in a model optimized for specific test data. This would lead to biased evaluation in assessing the generated embeddings and would no longer provide an effective evaluation of the model's ability to generalize to unseen data. Therefore, the idea is to use data that is known to be negative triples: the inconsistent triples generable through ontology axioms and the training set.

If the embeddings were optimal, all inconsistent negative triples would be correctly classified as such. If some negative triples are classified as positive, it indicates that their embeddings could be improved. Thus, these misclassified negative triples will be used as negative examples in the next iteration of training. The basic concept to "correct" the embeddings that were not learned properly was inspired by the paper "Improving Knowledge Graph Embeddings with Ontological Reasoning" [16] already presented in Chapter 3 in which a different procedure for this type of "correction" is employed. In the paper, the authors proposed using a link prediction method to identify triples that the model perceived as positive but were, in reality, inconsistent (and thus negative). In TransHI, however, a decision was made to opt for a classification task for the following reasons:

- By focusing solely on inconsistent triples, it becomes much easier to identify a large number of misclassified triples. Jain et al., for instance, had to employ a dedicated component to increase the number of inconsistent triples discovered post link prediction. In the case of TransHI, this step is unnecessary.
- Using link prediction, the task also produces many predicted triples. However, if the initial embeddings are of good quality, the majority of these triples will be consistent, rendering them unhelpful for the continuation of the pipeline.
- From an interpretative perspective, the classification algorithm is straightforward, elucidating the rationale behind labeling a triple as positive or negative. Conversely, understanding the underlying reasons for predictions made through link prediction can be more intricate.

This "training data update" step can be divided into two blocks: inconsistent triples generation and the classification algorithm.

- **Inconsistent Triples Generation.** The generation of inconsistent triples is done by finding a negative triple for each positive triple in the training set exactly in the same way used during training to create "ontology-based negative triples"(presented in section

4.4.1). This ensures that when the misclassified triples are used as negatives in the next iteration of training, they will be matched to the positive triples from which they were generated. In Chapter 4.4, particularly in pseudocode 4.3, it is shown that in each training epoch, each positive triple is used on average once. This determines how many inconsistent triples need to be generated for each positive. In the worst case scenario, if all the inconsistent triples produced for a positive triple were misclassified, in order to use them all in the next training iteration, the number of inconsistent triples should be equal to the number of training epochs. Therefore, whenever possible, a number of inconsistent triples equal to the epochs of training are generated for each positive. It should be noted that the generation algorithm ensures that the generated inconsistent triples are unique both within each iteration and across different iterations of the pipeline. Thus, at a certain iteration, it becomes no longer possible to generate new inconsistent triples, leading to the iterative pipeline getting blocked. The inability to generate more inconsistent triples is one of the factors leading to the termination of the pipeline.

- **Classification Algorithm.** Classification is done using a very simple standard algorithm that follows the simple procedure outlined in pseudocode 4.6. Using the embeddings generated from the previous training, the score for each triple to be classified is calculated. The score of triples is calculated exactly using the score function used during training:

$$f(e_h, e_r, e_t) = \|e_h + e_r - e_t\|_2^2$$

By comparing whether the score is smaller or larger than a threshold  $\delta$  specific to each relation, the triple is classified as positive or negative (line 12,13 in Alg4.6). The crucial aspect of this algorithm is the value that  $\delta$  should take for each relation. This is determined by iterating over all triples in the training set and setting  $\delta$  for each relation to be equal to the score farthest from zero among the scores of triples containing that specific relation (lines 5-10 in Alg.4.6). This is a simple and intuitive method to determine the threshold, also used by the authors of "Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs"[10] although they employed triple classification as a means to assess the quality of the embeddings. Using different thresholds for each relation is a strategy adopted because triples with different relations could have slightly different scores: some triples might have higher scores than others due to the diversity of information contained in the embeddings. In other words, using different thresholds takes into account the specific characteristics of each relation, leading to more accurate classification results.

---

**Algorithm 4.6** Classification Algorithm Pseudocode
 

---

```

1: Read embeddings from file
2: Read train triples and save them in trainList
3: Read inconsistent triples and save them in inconsistentList
4: Initialize the vector  $\delta[]$  of the thresholds with default values
5: for triple  $(h, r, t) \in \text{trainList}$  do
6:    $\#ABS\text{SCORE}(h, r, t) = |\text{SCORE}(h, r, t)|$ 
7:   if  $ABS\text{SCORE}(h, r, t) > \delta[r]$  then
8:      $\delta[r] = ABS\text{SCORE}(h, r, t)$ 
9:   end if
10: end for
11: for triple  $(h, r, t) \in \text{inconsistentList}$  do
12:   if  $ABS\text{SCORE}(h, r, t) < \delta[r]$  then
13:     Output  $(h, r, t)$  as misclassified triple
14:   end if
15: end for

```

---

The next step after identifying the misclassified triples is to use these triples as negatives in the next iteration of training. If no misclassified triples are found, the algorithm terminates. This, along with the inability to generate new inconsistent triples, constitutes the two reasons for the termination of the iterative pipeline.

## 4.6. TransHI - Training in next iterations

After performing the classification step and identifying misclassified inconsistent triples, it is possible to proceed with new iterations.

The embeddings for iterations subsequent to the first are not randomly initialized but instead use the embeddings generated from the previous iteration.

The main goal of iterations after the first is to correct embeddings that have not been learned optimally in previous iterations.

For simplicity, embeddings are updated in a manner consistent with the first iteration, utilizing pairs of positive and negative triples. The score function used is also exactly that of the first iteration (and of transE). As positive triples, the training triples are re-used. Therefore, it is preferable to avoid reusing the same positive-negative pairs from the initial iteration to avoid overfitting. Thus, the approach to generate negatives for subsequent iterations diverges from the first iteration due to the following considerations:

1. There's a necessity to incorporate the misclassified inconsistent triples as negative exam-

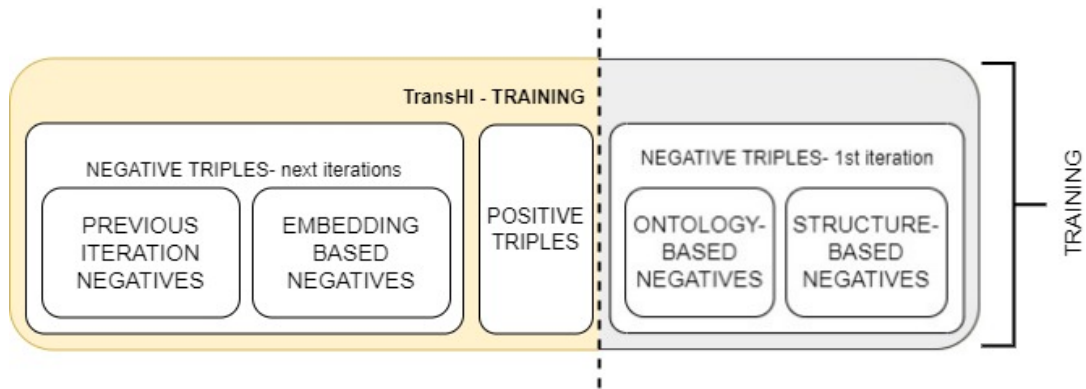


Figure 4.13: Schema of the TransHI component for the next iterations training.

ples.

2. Reusing triples generated with entity neighbors would lead to overfitting, especially since these triples were already employed during the initial iteration.
3. Generating ontology-based negative triples as in the first iteration is not advisable. The rationale behind this is that producing triples which the model has previously encountered and adapted to raises overfitting concerns. Moreover, if there was information from the ontology not learned adequately, it would be contained in the misclassified triples, given they are synthesized in the exact same manner as the first iteration's "ontology-based" triples.

The objectives of iterations after the first are as follows:

1. "Correct" the embeddings of misclassified triples by using them as negatives.
2. Avoid overfitting by not using triples that were used in previous iterations.
3. Avoid degrading embedding quality by using false negatives.

The pseudocode 4.7 outlines the functioning of the algorithm for iterations following the first and Fig. 4.13 represents the training component highlighting the part related to the iterations following the first. To achieve the delineated objectives, based on empirical results, it was decided to use two types of negative triples:

- Misclassified inconsistent triples from the previous iteration.
- Randomly generated triples with two constraints to avoid degrading embedding quality.

The mechanisms for generating triple negatives in the current iteration differ from those in the first iteration of TransHI. Consequently, the loss function is also expressed differently than the

one presented in formula 4.22 in section 4.4. It consists of two terms, one related to random triples and the other to triples generated in the previous iteration:

$$\begin{aligned}
 L = & \sum_{(h,r,t) \in S_{\text{Prev\_Iter}}} \sum_{(h',r,t') \in S'_{\text{Prev\_Iter}}} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'})) \\
 & + \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'} \max(0, \gamma + f(e_h, e_r, e_t) - f(e_{h'}, e_r, e_{t'}))
 \end{aligned} \tag{4.24}$$

For the first type of negative triples the inconsistent triples were created from training triples, as explained in section 4.5. It is sufficient to match the misclassified inconsistent triple to its corresponding training triple and use it as its negative (line 8 and 9 in Alg.4.7). Each such negative is used only once (line 10).

Given that the misclassified triples are created in the same manner as the ontology-based triples from the first iteration, they will not contain entities that do not belong to any class. For this reason, just as in the first iteration, the misclassified triples are used in a quantity inversely proportional to the percentage of entities that are not instances of any class (denoted as *pct\_no\_class\_entities* in Alg. 4.7).

For all triples that don't have corresponding misclassified triples (or they have all been used), a random approach is maintained to train entities uniformly and at the same time reduce the likelihood of reusing many of the triples used in the first iteration. Experimental evidence has demonstrated that relying solely on positive triples that have a corresponding misclassified negative could lead to an imbalanced training regimen. Entities or relations with erroneous embeddings are inclined to feature prominently in many misclassified triples, whereas those with accurate embeddings might not appear at all. Consequently, training exclusively on misclassified triples may result in overfitting to certain entities or relations while entirely neglecting others. Such an approach, with some entities or relations being overrepresented, could compromise the overall quality of the embeddings.

To avoid generating false negatives, a constraint is applied to randomly generated triples. This constraint is based on the idea that negative triples should have scores farther from zero with respect to positive triples' score. Thus, the constraint consists in accepting the randomly generated triple (by randomly corrupting either the head or tail entity) only if its relative score is farther from zero compared to the corresponding positive triple (line 16). This constraint was empirically tested and was inspired by the definition of the score function, which is meant to be  $\approx 0$  for positive triples and far from 0 for negative triples.

A second constraint is applied to these randomly generated triples. It is based on the intuition that to create a likely negative triple, the entity to be corrupted can be replaced with an entity that is very different from it. Therefore, the second constraint involves measuring the similarity between the entity to be corrupted and the randomly chosen replacement entity (lines 16 and

---

**Algorithm 4.7** TransHI Training Function for Following Iterations Pseudocode

---

```

def transHI_training_function_next_Iter (previous_embeddings, misclassified_triples)
1: Initialize embeddings with previous_embeddings
2: for  $i = 1$  to trainTimes do
3:   if  $\text{size}(\text{misclassified\_triples}) == 0$  then
4:     Break
5:   end if
6:   for  $j = 1$  to  $\text{size}(KG)$  do
7:     Sample positive triple  $(h, r, t)$  from  $S$ 
8:     if there exists triple  $m$  in misclassified_triples matching to  $(h, r, t)$  and
        $\text{rand\_num}(0,100) > \text{pct\_no\_class\_entities}$  then
9:       Negative triple  $(h', r, t') = m$ 
10:      Remove  $m$  from misclassified_triples
11:       $\text{updateEmbeddings}(h, r, t, h', r, t')$ 
12:    else
13:      if  $\text{rand\_num}(0,100) > \text{pct\_no\_class\_entities}$  then
14:        repeat
15:          Negative triple  $(h', r, t') = \text{randomNegative}(h, r, t)$ 
16:          until  $\text{score}(h, r, t) < \text{score}(h', r, t')$  and  $\text{cosineSimilarity}(h, h') < 0$ 
17:        else
18:          repeat
19:            Negative triple  $(h', r, t') = \text{randomNegative}(h, r, t)$ 
20:            until  $\text{cosineSimilarity}(h, h') < 0$ 
21:          end if
22:           $\text{updateEmbeddings}(h, r, t, h', r, t')$ 
23:        end if
24:      end for
25:    end for
26:  Output embeddings

```

---

20). This similarity is calculated using cosine similarity between the entity embeddings. Cosine similarity measures the cosine of the angle between two vectors, in this case, embeddings. Given two embeddings  $e_A$  and  $e_B$ , their cosine similarity is calculated as follows:

$$\text{COSINESIMILARITY}(\mathbf{e}_A, \mathbf{e}_B) = \frac{\mathbf{e}_A \cdot \mathbf{e}_B}{\|\mathbf{e}_A\|_2 \times \|\mathbf{e}_B\|_2}$$

Where  $\mathbf{e}_A \cdot \mathbf{e}_B$  is the scalar product of the embeddings and  $\|\mathbf{e}_A\|_2$  is the L2-norm of the  $e_A$  embedding. Cosine similarity ranges between 1 and -1:

- Cosine Similarity =1 indicates that the two entities are identical.
- Cosine Similarity =0 indicates that the two embeddings are orthogonal and the entities are likely unrelated.

- Cosine Similarity  $= -1$  indicates that the entity embeddings are diametrically opposite, and the entities are somewhat "opposite" to each other.

If entities have similar embeddings, with cosine similarity close to 1, it means they are semantically similar. Using a similar entity as a corruption for another similar entity might lead to false negatives. For this reason, it is preferred that the cosine similarity is low. The threshold for accepted similarity was found empirically and set to 0. Setting a lower threshold would further limit the similarity between entities, which can be positive but also limits the pool of selectable entities. This could lead to overfitting on these entities, so it is better to keep the range of selectable entities as wide as possible. However, allowing a cosine similarity  $> 0$  is risky as it implies some similarity between the two entities. Given that the entities with a similarity score less than 0 relative to a given entity are limited, especially when there is a large number of entities that do not belong to any class, there is a need to diversify the range of entities used to avoid overfitting too quickly. For this reason, the percentage of triples upon which the cosine similarity constraint is applied is set to be inversely proportional to the percentage of entities that are not instances of any class ('if' condition at line 13).

Empirically, it has been found that combining the two constraints presented yields better results compared to using them individually or not using them at all.

An important aspect of training in these subsequent iterations is that they don't necessarily run for all possible epochs: training continues as long as there are misclassified inconsistent triples from the previous iteration to use ('if' condition at line 2 in Alg.4.7). The main goal of these iterations is to "correct" the embeddings of misclassified triples. Without those, if training were to continue, it would only increase the possibility of overfitting as entities with similarity  $< 0$  to another entity are limited. Moreover, random generation is still subject to producing false negatives and/or triples that are not particularly useful for learning.

At the end of the second training, new embeddings are obtained. These can be used in a new classification step exactly like the first step. The only difference is that, as mentioned, the inconsistent triples to classify will be different to avoid overfitting. After classification, if misclassified triples are found, a new training is carried out, and this process continues iteratively.

As previously discussed in chapter 4.5, the two termination conditions for the pipeline are:

- The absence of further generable inconsistent triples to conduct the classification task. It is important to recall that this step is executed in every iteration using different inconsistent triples, and at a certain iteration, the generable triples are exhausted.
- The absence of misclassified triples during the classification. In this scenario, there would be no embeddings to "correct" since all classifications have been performed accurately.



Upon the completion of the TransHI iterations, one can compare the quality of the embeddings obtained across various iterations and choose to use those of the highest quality for the preferred application context.



# 5 | Experiments & Evaluation

In each iteration of the pipeline, novel embeddings are generated. The ultimate objective is to select the most optimal embeddings from one of these iterations.

These embeddings are evaluated for link prediction purposes. Two distinct metrics are employed for this assessment: mean rank and Hits@10 , which are elaborated upon in section 5.2.

The evaluation was conducted on three distinct real-world KGs, each possessing unique characteristics: DBPEDIA15k, YAGO, and NELL. The performance of TransHI is juxtaposed with that of TransE, TransOWL, and two variations of TransHI that exclusively use either the structure-based negatives or the ontology-based negatives. This comparative analysis provides a thorough perspective on the method’s performance, as well as the value of the hybrid approach that has been employed.

In this section, the technical specifications of TransHI’s implementation and the experimental settings are delineated. This is followed by a description of the link prediction algorithm and the corresponding metrics. Subsequent segments detail the experimental results for TransHI and draw comparisons with the aforementioned algorithms. These results are structured into different subsections, corresponding to the various stages of TransHI: a section for the assessment of the preprocessing phase (5.4), one focused on the initial training of embeddings (5.5), and another addressing the iterative component and subsequent training iterations (5.6).

## 5.1. Implementation and experimental settings

### Hardware and Software

Experiments were conducted using a computational cluster with the following specifications:

- Number of Machines: 16
- Model: Dell R610 (April 2011)
- Processor: Dual Intel Xeon with 8 cores each
- Memory: 24 GB RAM
- Operating System: Ubuntu 18.04

The implementation of TransHI was developed using a variety of programming languages: Java 1.8.0, Python 3.x, and C++ (compliant with the C++11 standard, compiled using g++ 6.3.0 from MinGW). The version of the HermiT reasoner used to work with ontologies is HermiT 1.3.8 <sup>1</sup>.

### Implementation Details

Several algorithms were adapted from the pre-existing code provided by the authors of TransOWL [10]<sup>2</sup>. The subsequent sections delineate the implementation specifics of each step in TransHI:

- **Inconsistency Correction:** code written from scratch developed in a Java environment, leveraging the HermiT reasoner for inconsistency checks.
- **Axiom Entailment:** code written from scratch in Java using the HermiT reasoner for exporting axioms from the ontology into files.
- **Positive Triples Augmentation:** code written from scratch in a Java environment.
- **Training Algorithm:** Code written in C++ by adapting the TransE algorithm from the authors of TransOWL. Components pertaining to the score function and embedding updates remained unchanged. Additions include file importation features (incorporating ontological axioms, neighbors, and entity CCs), and the integration of functionalities for the construction and utilization of negative triples during training.
- **Inconsistent Triples Generation:** code developed from scratch in C++, chosen for its computational efficiency.
- **Triple Classification:** code developed in Python, adapting the triple classification algorithm provided by the authors of TransOWL. Modifications were made concerning the management and output of misclassified triples.

Supplementary codes accompanying the main ones include:

- **Neighbors Finder:** a code written from scratch in Java that, given a KG, conducts  $\omega$  random walks to identify k-hop neighbors for every entity in the KG, with  $\omega$  and k serving as hyperparameters.
- **CC Generator:** code written from scratch in Java that, upon receiving a KG, determines the CC associated with all the entities in the graph.

---

<sup>1</sup>The link to the official HermiT website is provided: "<http://www.hermit-reasoner.com/java.html>"

<sup>2</sup>The GitHub repository of the authors can be accessed at: "<https://github.com/Keehl-Mihael/TransROWL-HRS>"

Lastly, the link prediction task's code, used for embedding evaluations, is written in C++ and mirrors the link prediction algorithm furnished by the authors of TransOWL. The source code adopting the TransE approach is available on a dedicated GitHub repository<sup>3</sup>.

### Parameters settings

Concerning the parameters requisite for training embeddings, standard parameters commonly employed in the literature were used, also referenced in [10], namely:

- Learning rate ( $\alpha$  in formula 4.23): 0.001
- Epochs ("TrainTimes" in pseudocodes 4.3,4.7): 1000
- Embedding dimension: 100
- Margin  $\gamma$ : 1

Maintaining the same parameters as those used by the authors of TransOWL enables a comparison between TransOWL and TransHI under identical conditions. Two additional parameters, associated with the training employed by TransHI, pertain to the process of identifying entity neighbors. A choice of  $k = 3$  hops was made, and varying numbers of random walks were used for the three KGs:

- DBPEDIA15k: 10,000 random walks;
- YAGO: 3,000 random walks;
- NELL: 5,000 random walks;

These choices were informed by the quantity of neighbors required for training and will be discussed in section 5.5.

## 5.2. Link prediction and evaluation metrics

To assess the quality of the produced embeddings a link prediction algorithm is used. This procedure is employed both to ascertain the optimal embeddings achievable in one of the TransHI iterations and multiple times during the initial iteration. As highlighted in Section 4.4, the first iteration of TransHI necessitates an empirical determination of the ideal percentage for generating negative triples at random. Consequently, the objective is to identify the percentage that yields the highest quality embeddings.

---

<sup>3</sup>The link to the new framework's GitHub repository is provided: "<https://github.com/Elisamariani12/TransHI>"

---

**Algorithm 5.1** Link Prediction Pseudocode

---

```

1: for (h,r,t) ∈ test_triples do
2:   score_test = score(h,r,t)
3:   Initialize two ordered ranking lists: ranking_H, ranking_T
4:   for each entity in KG do
5:     if entity != h then
6:       score_substitute = score(entity,r,t)
7:       Add (entity,r,t) to ranking_H based on score_substitute
8:     end if
9:     if entity != t then
10:      score_substitute = score(h,r,entity)
11:      Add (h,r,entity) to ranking_T based on score_substitute
12:    end if
13:    Update MR and H@10
14:  end for
15: end for
16: Output final MR and H@10

```

---

As previously outlined in Section 2.4.2, link prediction algorithms predict the existence of a connection between two entities by calculating the probability of missing edges. This prediction of missing links can be done based on existing embeddings, and that's precisely what is done in this case. By predicting missing links using embeddings and comparing them to the real links in the KG (i.e., the test data), it is possible to assess the quality of the embeddings. The pseudocode 5.1 outlines the functioning of the implemented link prediction algorithm.

In particular, for each triple in the test set, its score is calculated (line 2 in Alg.5.1). Then, both the head and tail of that triple are substituted with all other entities in the dataset, and their scores are calculated as well (lines 6 and 10). The scores are computed using the same score function used during training and classification:

$$f(e_h, e_r, e_t) = \|e_h + e_r - e_t\|_2^2$$

. The closer a triple's score is to zero, the more probable it is. After calculating all possible scores by changing the head or tail entity of the test triple, there's the option to "filter out" triples that were already part of the training set, resulting in "filtered" results. This procedure can be done when the focus is more on the model's ability to predict new links rather than having an overall view of the model's performance. Since the objective of this step is to evaluate the overall quality of the embeddings, the evaluation phase of the framework will focus on the "unfiltered" results.

Once the scores are calculated and potentially filtered, the model's performance can be evaluated by comparing the scores of the true test triples against the others. If a test triple has a low score, it means the model has successfully predicted it as a likely triple. To evaluate how well

the test triples were predicted as likely, two metrics can be used:

- **Mean Rank (MR)**: After ordering the scores of generated triples for each test triple in ascending order, the average rank of the scores of those test triples is calculated. A low MR is preferable as it means the test triples tend to have lower scores.
- **Hits@10 (H@10)**: This metric represents the percentage of correct test triples that are among the top 10 triples with the lowest scores in their respective rankings. Having a higher H@10 percentage means the correct test triples were predicted as very likely.

In light of the evaluation relying on two indexes, the MR and H@10, the comparison between two sets of embeddings gains complexity. Specifically, when one set of embeddings displays a lower MR while another set exhibits a higher H@10, discerning the superior option becomes intricate. Notably, these two metrics assess distinct facets of the model's performance, which could potentially yield conflicting insights into the preferred model

The evaluation of the embeddings will be based on the MR and H@10 indices. The comparison could become complex in a case where one set of embeddings displays a lower MR while another set exhibits a higher H@10: discerning the superior option becomes intricate. The two discussed metrics measure different aspects of the model's performance, so they might provide conflicting information about which model is better. To use a single evaluation criterion, the decision was made to use the ratio  $\frac{H@10}{MR}$ . This allows for an evaluation that combines the information from both metrics: its value increases if H@10 increases and/or MR decreases, and decreases if H@10 decreases and/or MR increases. Therefore, this ratio is used to select the set of embeddings with optimal performance produced by the framework.

By using this link prediction algorithm and the H@10 and MR metrics, a comprehensive view of the embeddings' quality is obtained. Having a low MR and a high H@10 ensures that the embeddings are of high quality and effectively represent the semantic information in the KG.

### 5.3. Overview of the three datasets

The characteristics of the KGs analyzed are now examined, as they are crucial for the interpretation of the experimental results. These datasets, DBPEDIA15k, YAGO, and NELL, have been previously introduced in Section 2.2.4. Table 5.1 annotates the primary features of these databases: number of entities, number of relations, density, total graph size, and the division of triples into training, testing, and validation datasets. It should be noted that to maintain comparability with the performance achieved by the authors of TransOWL, the division between testing, training, and validation datasets was preserved as they used, even though only the training and testing sets are actually be utilized within the framework.

A clarification is needed regarding the ontology of YAGO. Instead of sourcing the ontology

	DBPEDIA15k	YAGO	NELL
<b>Size KG</b>	183218	265285	148437
<b>n°entities</b>	12862	87795	68620
<b>n°relations</b>	276	316	272
<b>density</b>	0,0011076	3,44174E-05	3,15244E-05
<b>% train</b>	70	80	80
<b>% test</b>	20	10	10
<b>% validation</b>	10	10	10
<b>n° ont.axioms</b>	26699	1122106	296580

Table 5.1: Features of the KGs used for TransHI's evaluation.

from a single file on YAGO's official website, a compilation was made by combining various "pieces" of ontology downloaded from the same site<sup>4</sup>. Specifically, all pieces of the ontology that were deemed useful for the framework and for inconsistency detection were amalgamated. This step was necessitated by the expansive size of the complete YAGO ontology, which includes components like annotations and other details that, although relevant in other scenarios, were not pertinent within the scope of the framework. Therefore, a streamlined version of the ontology was utilized, aligning with the pipeline application requirements.

## 5.4. Preprocessing results

### Inconsistencies Correction

The table 5.2 shows the number of triples before and after the "Inconsistencies Correction" phase in the three KGs.

	DBPEDIA15k	YAGO	NELL
<b>Size KG - pre correction</b>	183218	265285	148437
<b>Size KG - post correction</b>	151442	265285	104124
<b>% preserved triples</b>	83%	100%	70%

Table 5.2: Results of the "Inconsistencies Correction" step.

As seen from the table, DBPEDIA15k and NELL undergo correction, and a significant portion of triples is removed due to being inconsistent. It is evident that in NELL, more triples are removed than in DBPEDIA15k even though the initial KG size is smaller. This can be justified

<sup>4</sup>Official YAGO website: "<https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/>"



by examining the size of their respective ontologies: NELL's ontology is considerably richer. Regarding YAGO, despite its large ontology, it remains consistent with it.

After removing inconsistencies from the three KGs, they are split into train, test, and validation sets, according to the percentages already shown in table 5.1.

### Ontology Axioms Entailment

The quantities of extracted axioms, including the ones contained in the three ontologies related to the KGs and entailed from them, are represented in table 5.3.

Axioms	DBPEDIA15k	YAGO	NELL
<b>DisjointWith</b>	277	0	18679
<b>Domain</b>	744	78720	3850
<b>Range</b>	1164	33834	3001
<b>Equivalent classes</b>	66	66	0
<b>SubClassOf</b>	139	17464	647
<b>SubPropertyOf</b>	8	0	0
<b>Equivalent properties</b>	0	37	0
<b>Inverse properties</b>	0	4	0
<b>Asymmetric properties</b>	0	0	239
<b>Irreflexive properties</b>	0	0	268

Table 5.3: Axioms contained in and entailed from the three ontologies.

As observed from the table, these three ontologies contain significantly different types of axioms, which will impact both the creation of new positive triples and the training. It should be noted that any types of axioms mentioned in the framework presentation that are not present in this table are absent because they were not present in the ontologies.

### Positive Triples Augmentation

Regarding the new positives that can be created using the axioms, table 5.4 reports the quantities of triples added to the training triples and also breaks down the distribution between "typeOf" triples (with "type" relation) and "noTypeOf" triples.

As can be seen, for NELL no "noTypeOf" triples are produced. This is due to the fact that, as shown in table 5.3, the axioms leveraged in the "augmentation" step to generate triples of this kind (axioms 4.14-4.21, previously introduced in section 4.3.3) are not present in the NELL ontology. Instead, the number of "typeOf" triples created is quite large, exceeding the size of the initial training set. It should be noted that the triples augmentation step, which is based

on ontological axioms, already infuses the KG with additional information derived from the ontology. In the case of NELL, this step significantly enriches the KG with such data.

	DBPEDIA15k	YAGO	NELL
<b>Size Train set - pre augmentation</b>	106178	212228	83710
<b>n° added triples</b>	17351	27631	88764
<b>n° added triples - typeOf</b>	17066	23618	88764
<b>n° added triples - noTypeOf</b>	285	4013	0
<b>Size Train set - post augmentation</b>	123529	239859	172474
<b>% added triples</b>	16%	13%	106%

Table 5.4: Results of the "Positive Triples Augmentation" step.

## 5.5. Training - first iteration results

Once the preprocessing phase is completed, the three training sets can be used to produce embeddings. The quality of these embeddings is determined using the link prediction algorithm introduced in section 5.2. To provide a comprehensive overview of the method's performance, the training results obtained with the link prediction task with embeddings generated with TransHI are compared with:

- TransE (original data): TransE algorithm applied to non-preprocessed KGs;
- TransOWL (original data): TransOWL algorithm applied to non-preprocessed KGs;
- TransE (preprocessed data): TransE algorithm applied to preprocessed KGs;
- TransOWL (preprocessed data): TransOWL algorithm applied to preprocessed KGs;
- TransHI-structure based only: the algorithm obtained by using only the structure-based negative triples from TransHI. This algorithm is applied to preprocessed KGs;
- TransHI-ontology based only: the algorithm obtained by using only the ontology-based negative triples from TransHI. This algorithm is applied to preprocessed KGs;

Specifically, regarding the results of TransE and TransOWL on original data, these are similar with those obtained by the authors of TransOWL [10]. By applying TransE and TransOWL to the preprocessed data, not only the effectiveness of the whole framework but also the individual contribution of the training phase to the quality of the embeddings can be evaluated.

As already explained in section 4.4, to determine the ideal percentage of randomly generated negative triples relative to ontology-based and/or structure-based triples, it is recommended to

try different percentages. The tables presented in this section display the results obtained with the best percentage of random triples, and Appendix A shows the results for percentages ranging from 0 to 100, in increments of 10.

Regarding the neighbors used to obtain the results for TransHI and the TransHI version using only structure-based triples, they were chosen with the number of hops  $k = 3$ . This is because, as highlighted by the authors of SANS [1] and discussed in section 3.2, "the semantic meaningfulness of SANS-based negatives declines as the size of the  $k$ -hop neighbourhood" is increased. Thus, the authors recommend choosing a relatively low value for  $k$ . Experimentally, it was found that using  $K=2$  did not provide enough neighbors to be used throughout the training, since, as explained in section 4.4.2, neighbors related to one single entity are used once to diversify the range of used entities. Therefore,  $K=3$  was chosen, as increasing the hop number exponentially increases the number of neighbors. Using this number allowed for a larger quantity of neighbors, which were enough for use throughout all training epochs. At the same time, increasing  $K$  by 1 did not compromise the effectiveness of the negative triples formed with such neighbors: as will be seen from the results, they were sufficiently meaningful to aid learning. Using an insufficient number of neighbors for the construction of negatives throughout the learning process has not only proven to be empirically less effective, but also undermines one of the foundational ideas of TransHI: to compensate for the limitation that ontology alone cannot generate negative triples corresponding to every positive triple by creating equally meaningful negatives using the structure of the KG.

Regarding the number of random walks performed to obtain such neighbors, this was determined by time constraints: the higher the number of entities in a KG, the longer it takes to compute the  $k$ -hop neighbors for the entire training set. Therefore, for DBPEDIA15k, 10.000 random walks were used, for YAGO 3.000 and for NELL 5.000. After finding these neighbors, it was verified that they were sufficient for generating negatives throughout all training epochs.

Concerning the training results presented in the forthcoming tables, the evaluation of MR and H@10 (introduced in 5.2) was conducted both generally, on all test data, and by separately evaluating triples of the type "typeOf" and "noTypeOf." Evaluating only triples of the type "typeOf" or "noTypeOf" means measuring the ability of the embeddings to represent such types of relations. This distinction is very popular in the literature as it allows for exploring how well the embeddings represent the association of entities to their respective classes and other relations. Specifically, as already discussed in section 5.2, the choice of the optimal percentage of randomly generated negative triples is made by looking at the highest H@10/MR value, as it increases when MR is low and H@10 is high, which is precisely what would be desirable.

To better understand and interpret the results, the percentages of entities that are not instances of any class and are therefore not involved in any triples of the type "typeOf" are also reported. Specifically, the percentages of non-class entities among the training entities are the following:

	DBPEDIA15k	YAGO	NELL
% entities without class	29	86	59

Table 5.5: Percentages of entities without an associated class.

Below are the individual results from the first iteration of TransHI and the aforementioned algorithms across the three databases, accompanied by pertinent observations.

#### DBPEDIA15K

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	587,07	32,46	692,29	9,75	1279,36	42,21	0,0329931
TRANS- OWL (original data)	621,06	32,24	493,46	13,20	1114,52	45,44	0,0407709
TRANS-E (preprocessed data)	611,63	33,29	295,13	12,78	906,76	46,07	0,0508073
TRANS- OWL (preprocessed data)	641,76	33,72	287,89	12,71	929,65	46,43	0,0499435
TransHI-ontology based only	588,23	33,37	294,87	12,66	883,10	46,03	0,0521232
TransHI-structure based only	581,74	33,41	284,46	12,79	866,20	46,20	0,0533364
<b>TransHI</b>	<b>577,47</b>	<b>33,15</b>	<b>284,33</b>	<b>12,89</b>	<b>861,80</b>	<b>46,04</b>	<b>0,0534231</b>

Figure 5.1: DBPEDIA15K: TransHI's first iteration results (better = green, worse = red).

As can be seen from the first four rows of the table in Fig. 5.1, the pre-processing phase has significantly improved the performance of the TransE and TransOWL algorithms: the MR values are much lower, and the H@10 scores are considerably higher when the algorithms are run with pre-processed data. This is primarily attributable to an improvement observed in the predictions of "typeOf" triples. The significant enhancement in these predictions can be explained by the fact that 98% of the triples added during the "Positive Triples Augmentation" phase were of the "typeOf" type.

Analyzing the "TransHI-ontology based only" row, it can be noticed that the use of ontology-based negative triples greatly enhances learning compared to TransE and TransOWL: the H@10/MR ratio is significantly higher. With structure-based negatives, even more remarkable results are achieved. Improvement is also noticeable when structure-based triples are combined with ontology-based triples: the best version corresponds to TransHI. In this KG, both the effectiveness of the pre-processing performed and the effectiveness of TransHI compared to TransE and TransOWL are evident and the results improve not only from those of TransE and TransOWL on the original data but also compared to those on the pre-processed data.

The tables containing all the results obtained using various percentages of randomly generated triples, from which the best ones were extracted to compose table in Fig. 5.1, are found in Appendix A (Figures A.1,A.2,A.3).

## YAGO

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	7417,08	19,24	587,19	8,71	8004,27	27,95	0,0034919
TRANS- OWL (original data)	7455,49	19,21	580,29	8,68	8035,78	27,89	0,0034707
TRANS-E (preprocessed data)	7437,56	19,68	466,28	9,13	7903,84	28,81	0,0036451
TRANS- OWL (preprocessed data)	7292,21	19,84	473,94	9,04	7766,15	28,88	0,0037187
TransHI-ontology based only	7343,10	19,68	470,36	9,19	7813,46	28,87	0,0036949
TransHI-structure based only	7013,95	20,15	473,77	9,25	7487,72	29,40	0,0039264
<b>TransHI</b>	<b>7020,42</b>	<b>20,31</b>	<b>472,94</b>	<b>9,15</b>	<b>7493,36</b>	<b>29,46</b>	<b>0,0039315</b>

Figure 5.2: YAGO: TransHI's first iteration results (better = green, worse = red).

In YAGO's results (5.2), a similar situation to that of DBPEDIA can be observed: the results of TransE and TransOWL improve on processed data compared to the original data. It should be remembered that in the case of YAGO, the KG was already consistent, and thus the "Inconsistencies Corretion" step was not needed. The improvement observed between the performance of TransE and TransOWL with pre-processed and non-pre-processed data is therefore entirely due to the "Triple Augmentation" phase. Specifically, as observed earlier, with the preprocessed data, there is a marked improvement in performance for "typeOf" triples. The reason aligns with that of DBPEDIA: the augmentation of positive triples predominantly targets triples of this kind, with 85% of all added being "typeOf".

Again, the algorithm that achieves the best results is TransHI. A difference that can be noted compared to DBPEDIA15K is that ontology-based triples, when used alone, provide better results than TransE and comparable to those of TransOWL, but significantly lower compared to TransHI. This is likely due to two reasons:

- In YAGO, entities without a class make up 86% of the total entities (as seen in Table 5.5). This means that ontology-based triples are created using only 14% of the entities, which does not allow for uniform learning;
- As seen in Table 5.3, no axioms are extracted for YAGO that allow for the creation of ontology-based triples of the "typeOf" kind (reference is made to Chapter 4.4.1 where it is explained which axioms are used for the creation of which negative triples). The inability to create this type of negative triples also impacts the uniformity of learning across entities and relations.

YAGO thus demonstrates the importance of also using another method of creating negative triples in addition to ontology-based ones, to "intervene" in cases where the latter are not generable and to create triples that are effective, meaningful, and assist learning, such as the structure-based ones used by TransHI.

The tables containing all the results obtained using various percentages of randomly generated triples, from which the best ones were extracted to compose table in Fig. 5.2, are found in Appendix A (Tables in Fig.A.4,A.5,A.6).

### NELL

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	7162,08	19,01	2872,45	6,55	10034,53	25,56	0,0025472
TRANS- OWL (original data)	9622,40	15,54	2263,09	6,52	11885,49	22,06	0,0018560
TRANS-E (preprocessed data)	6112,15	19,79	3527,43	6,12	9639,58	25,91	0,0026879
TRANS- OWL (preprocessed data)	5730,69	19,89	4048,52	3,89	9779,21	23,78	0,0024317
TransHI-ontology based only	6026,96	19,61	3525,05	6,49	9552,01	26,10	0,0027324
TransHI-structure based only	5370,85	19,73	2622,63	8,56	7993,48	28,29	0,0035391
<b>TransHI</b>	5380,67	19,80	2557,24	8,05	7937,91	27,85	0,0035085

Figure 5.3: NELL: TransHI’s first iteration results (better = green, worse = red).

In NELL (table in Fig. 5.3), as in the first two KGs, the effectiveness of the data pre-processing phase is evident. In this instance, however, the predictions for "noTypeOf" triples witness a more substantial improvement. Even though the "Positive Triples Augmentation" phase produces 100% "typeOf" type triples in this context, NELL’s ontology is enriched with a greater abundance of domain and range axioms compared to those of DBPEDIA15k and YAGO. The presence of these axioms results in the creation of more "noTypeOf" negative triples, thereby enhancing their prediction. Another factor to note during the preprocessing of NELL is its higher inconsistency count compared to other datasets. By rectifying these inconsistencies, the generation of consistent embeddings for the entities is evidently facilitated.

Unlike the other two KGs, however, the combination of ontology-based triples with structure-based ones in this case is not the best option. The reason for these results lies in the pre-processing phase: in NELL, the training dataset had more than doubled. It expanded from 83,710 triples to 172,474 triples (by 106%). All these triples were created using ontological knowledge. Therefore, the KG used for training is already infused with ontological information, can lead to overfitting. As a matter of fact, the ontology-based triples may contain information on the equivalence of classes and subclasses that has already been abundantly introduced during the triple augmentation phase. In particular, unlike subsequent training phases where the focus is on ontological information that the model has not yet learned from previous iterations, in this initial iteration, ontology-based triples are simply generated using ontological axioms. This information is likely to be learned by the model anyway due to the presence of positive triples created leveraging the same ontological axioms.

In cases like this, where the "Positive Triples Augmentation" phase leads to having a large amount of ontological information already present in the positive triples, it makes sense that using another type of approach for the construction of negatives, such as the one using only

structure-based triples, leads to better results. The advice that can be given in these cases is therefore to replace in the pipeline, for the training of the first iteration, the hybrid algorithm of TransHI with another non-ontology-based learning algorithm.

It should be noted, however, that the results of TransHI even in this case are better compared to those of TransE and TransOWL, even on pre-processed data.

The tables containing all the results obtained using various percentages of randomly generated triples, from which the best ones were extracted to compose table in Fig. 5.3, are found in Appendix A (Figures A.7,A.8,A.9).

### General considerations on the results of the first iteration

The pre-processing phase demonstrated notable effectiveness, particularly in instances where the Knowledge Graph (KG) under examination exhibited a multitude of inconsistencies. Specifically, in DBPEDIA and NELL, the improvements were more pronounced than in YAGO. This is attributed to the fact that consistency checks in YAGO were rendered unnecessary as it already exhibited consistency with its corresponding ontology. Nonetheless, the case of YAGO provided valuable insights into the efficacy of the sole process of triple augmentation. This assertion is corroborated by the observation that both TransE and TransOWL exhibited enhanced performance on the pre-processed data, as opposed to the original data.

Concerning the efficacy of TransHI, in both DBPEDIA and YAGO — where the percentage of triples added during the "Positive Triples Augmentation" was relatively modest, at 16% and 13% respectively — the merits of adopting a hybrid approach were clearly evident. However, the case of NELL illuminated a potential shortcoming of the TransHI approach: when the training set is extensively augmented with ontological knowledge during the "Positive Triples Augmentation" phase (as seen in NELL with a 106% increase in triples), the utilization of ontology-driven negative triples can inadvertently instigate overfitting. In such scenarios, relying exclusively on structure-based triples proved more efficacious. Consequently, it is advised that for KGs which undergo substantial augmentation in the pre-processing phase, the inaugural training iteration should deploy an algorithm that favors non-ontology-based negative triples

## 5.6. Training Data Update & next iterations results

After executing TransHI with the optimal percentage of randomly generated triples, the steps of generating inconsistent triples and classifying such triples are carried out.

In all three KGs, as expected, several misclassified triples were obtained. This is because the embeddings are not perfect and do not capture all the semantic nuances that characterize the entities. These misclassified triples were utilized as negatives in the trainings of the subsequent iterations.



The results of the first 10 iterations conducted on each KG are presented below. For all three KGs, it was possible to complete more than 10 iterations; specifically, for DBPEDIA15k, the total iterations were 14, while for YAGO and NELL, they were over 50. Only the first 10 iterations are reported, as the iteration with the best quality embeddings, in all three cases, is found within these first 10 iterations.

In DBPEDIA15k (Fig.5.4), the best-quality embeddings were obtained in the fourth iteration; in YAGO (Fig.5.5), in the second; and in NELL (Fig.5.6), in the sixth. In all three cases, the performance starts to decline after the optimal iteration, which is due to the nature of the negative triples used in the iterations following the first.

In NELL, even though using just structure-based negative triples was preferable to TransHI for the first iteration, the subsequent iterations of TransHI are effective. Results achieved are notably superior to those obtained during the first iteration, even with respect to the ones achieved using just structure-based triples. This occurs because the ontological information used in subsequent iterations, unlike the one used in the first, is not redundant with that contained in the positive training triples. Utilizing misclassified triples, the information leveraged during training precisely corresponds to what the embeddings did not adequately capture in the initial training iteration.

As previously discussed in Chapter 4.6, the constraint that allows the use of random triples only if the entity replaced in the positive triple to create the negative one has a similarity  $< 0$  enables the generation of triples that are likely to be negative by exploiting the embeddings created in the previous iteration. It was also noted that the entities with a cosine similarity  $< 0$  to a given entity are limited. For this reason, after some iterations, the random triples generated through this constraint will begin to repeat, leading to gradual overfitting on these triples and a corresponding decline in the quality of the embeddings.

In Chapters 4.5 and 4.6, it was highlighted that the stopping conditions of the iterative pipeline are either the absence of more generable inconsistent triple files or the absence of misclassified triples. Realistically, since the performance begins to decline gradually due to overfitting after reaching the optimal iteration, the pipeline could be terminated a few iterations after the optimal one.

### General Considerations on Subsequent Iteration Results

The iterative approach demonstrated efficacy across all three examined KGs primarily because it concentrates on the enhancement of inadequately learned embeddings and imposes constraints, which, in the initial iterations, prevent any deterioration of other embeddings. However, the introduction of such constraints eventually results in the recurrent utilization of the training triples, leading to a subsequent decline in embedding quality after reaching its best performance. This progressive degradation, attributable to overfitting in iterations subsequent to the optimal one, implies that the pipeline might justifiably be terminated a few iterations post achieving the



optimal results.

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	587,07	32,46	692,29	9,75	1279,36	42,21	0,0329931
TRANS- OWL (original data)	621,06	32,24	493,46	13,20	1114,52	45,44	0,0407709
TRANS-E (preprocessed data)	611,63	33,29	295,13	12,78	906,76	46,07	0,0508073
TRANS- OWL (preprocessed data)	641,76	33,72	287,89	12,71	929,65	46,43	0,0499435
TransHI - 1st iteration	577,47	33,15	284,33	12,89	861,80	46,04	0,0534231
TransHI - 2nd iteration	560,80	32,81	283,60	12,94	844,40	45,75	0,0541805
TransHI - 3rd iteration	563,86	32,90	283,34	12,94	847,20	45,84	0,0541076
<b>TransHI - 4th iteration</b>	562,58	32,99	283,33	12,93	845,91	45,92	<b>0,0542847</b>
TransHI - 5th iteration	563,41	32,90	283,30	12,93	846,71	45,83	0,0541272
TransHI - 6th iteration	562,82	32,61	283,05	12,93	845,87	45,54	0,0538381
TransHI - 7th iteration	564,29	32,63	283,12	12,93	847,41	45,56	0,0537638
TransHI - 8th iteration	562,41	32,62	283,31	12,93	845,72	45,55	0,0538594
TransHI - 9th iteration	563,52	32,71	283,43	12,93	846,95	45,64	0,0538875
TransHI - 10th iteration	563,46	32,68	283,42	12,93	846,88	45,61	0,0538565

Figure 5.4: DBPEDIA15K: TransHI's first 10 iterations results (better = green, worse = red).

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	7417,08	19,24	587,19	8,71	8004,27	27,95	0,0034919
TRANS- OWL (original data)	7455,49	19,21	580,29	8,68	8035,78	27,89	0,0034707
TRANS-E (preprocessed data)	7437,56	19,68	466,28	9,13	7903,84	28,81	0,0036451
TRANS- OWL (preprocessed data)	7292,21	19,84	473,94	9,04	7766,15	28,88	0,0037187
TransHI - 1st iteration	7020,42	20,31	472,94	9,15	7493,36	29,46	0,0039315
<b>TransHI - 2nd iteration</b>	7040,06	20,34	472,78	9,22	7512,84	29,56	<b>0,0039346</b>
TransHI - 3rd iteration	7082,88	20,56	475,22	9,06	7558,10	29,62	0,0039190
TransHI - 4th iteration	7095,10	20,43	475,45	9,12	7570,55	29,55	0,0039033
TransHI - 5th iteration	7172,56	20,18	476,45	9,12	7649,01	29,30	0,0038306
TransHI - 6th iteration	7168,74	20,65	478,03	9,08	7646,77	29,73	0,0038879
TransHI - 7th iteration	7202,37	20,37	477,56	8,97	7679,93	29,34	0,0038203
TransHI - 8th iteration	7226,17	20,25	477,67	9,05	7703,84	29,30	0,0038033
TransHI - 9th iteration	7212,85	20,26	477,93	9,09	7690,78	29,35	0,0038163
TransHI - 10th iteration	7184,70	20,47	479,54	9,07	7664,24	29,54	0,0038543

Figure 5.5: YAGO: TransHI's first 10 iterations results (better = green, worse = red).

	NotTypeOf		TypeOf		General		
	MR	H@10	MR	H@10	MR	H@10	H@10/MR
TRANS-E (original data)	7162,08	19,01	2872,45	6,55	10034,53	25,56	0,0025472
TRANS- OWL (original data)	9622,40	15,54	2263,09	6,52	11885,49	22,06	0,0018560
TRANS-E (preprocessed data)	6112,15	19,79	3527,43	6,12	9639,58	25,91	0,0026879
TRANS- OWL (preprocessed data)	5730,69	19,89	4048,52	3,89	9779,21	23,78	0,0024317
TransHI - 1st iteration	5380,67	19,80	2557,24	8,05	7937,91	27,85	0,0035085
TransHI - 2nd iteration	5152,32	19,68	2641,13	7,60	7793,45	27,28	0,0035004
TransHI - 3rd iteration	5048,59	20,14	2671,61	8,44	7720,20	28,58	0,0037020
TransHI - 4th iteration	4958,95	20,32	2674,46	8,04	7633,41	28,36	0,0037152
TransHI - 5th iteration	4937,26	20,22	2653,01	8,25	7590,27	28,47	0,0037509
<b>TransHI - 6th iteration</b>	<b>4967,12</b>	<b>20,31</b>	<b>2679,67</b>	<b>8,59</b>	<b>7646,79</b>	<b>28,90</b>	<b>0,0037794</b>
TransHI - 7th iteration	5033,32	20,32	2727,75	7,70	7761,07	28,02	0,0036103
TransHI - 8th iteration	5040,42	20,38	2640,29	8,00	7680,71	28,38	0,0036950
TransHI - 9th iteration	5009,80	20,25	2647,78	8,60	7657,58	28,85	0,0037675
TransHI - 10th iteration	5121,79	20,27	2606,29	8,75	7728,08	29,02	0,0037551

Figure 5.6: NELL: TransHI's first 10 iterations results (better = green, worse = red).

# 6 | Conclusions and future developments

## 6.1. Contribution

The approach presented in this study, TransHI, introduces an optimized pipeline for generating high-quality embeddings suitable for numerical representations of KGs. Specifically, the preprocessing phase has proven to be especially effective when the inspected KG contained numerous inconsistencies: the inconsistency correction step was thus conducted using potent heuristics that allow for the efficient removal of inconsistent data within the KG while preserving as much information as possible. During the evaluation phase, especially with the KG YAGO, it was observed that even for consistent KGs, the preprocessing phase still enhances learning. Merely increasing the positive triples used in the training phase, during the "Positive Triples Augmentation" step, yielded results superior to those of the original datasets.

Turning to the results achieved through the first iteration of the newly implemented training algorithm, they underscore the efficacy of combining negative triples generated via ontology with those produced using the structural properties of the graph. However, one limitation of TransHI became evident. When the training set is heavily "infused" with ontological knowledge during the "Positive Triples Augmentation" preprocessing phase, using negative triples generated through ontology can lead to overfitting. As observed in Chapter 5, in the case of the NELL database, the training triples augmented by 106% during the "Positive Triples Augmentation" step. Relying solely on structure-based triples, yielded superior results in this case. The recommendation for KGs that undergo significant augmentation during the augmentation phase is to employ, for the first training iteration, an algorithm that uses non-ontology-based negative triples, such as SANS or the TransHI version that uses only structure-based triples. This approach prevents overfitting the ontological knowledge already present in the positive triples while ensuring the use of effective and beneficial triples for learning.

While iterative training for embedding generation is not an entirely novel concept and has been employed in various methodologies proposed in the literature, it has never been exploited using misclassified triples as negative examples for the subsequent training iteration. This approach proved to be successful across all three analyzed KGs. The constraints that TransHI imposes,

even on the generation of random triples, ensured that with subsequent iterations of the pipeline, the quality of embeddings that were not correctly learned could be improved without degrading the quality of embeddings not involved in such triples. It was observed that the introduction of these constraints, while being a commendable choice that enhances the quality of embeddings in the initial iterations, eventually leads to repeated use of the training triples, causing a gradual decline in embedding quality after reaching its peak. This incremental degradation due to overfitting in iterations subsequent to the optimal one suggests that the pipeline could reasonably be halted a few iterations after achieving the best results.

In fact, the theoretically possible iterations conclude when no more inconsistent triples are available for use during the classification step or when the latter classifies all test examples correctly. However, if the objective is to achieve optimal quality embeddings, it is reasonable to halt a few iterations after the best one, knowing that results will not improve in subsequent iterations due to overfitting concerns.

In summary, TransHI has demonstrated better performances than TransE and TransOWL algorithms for link prediction purposes, showcasing universal effectiveness across the studied cases. The hybrid approach underscored the significance of maintaining a balance between ontological knowledge and knowledge derived from the graph's structure. Should one prevail over the other, as in the case of training sets heavily infused with ontological knowledge, there exists a potential for overfitting. Nonetheless, such a risk is readily mitigable and does not compromise the effectiveness of the iterative component of the pipeline.

## 6.2. Future work

TransHI has demonstrated significant efficacy in representing knowledge graphs. However, there are further extensions and adaptations that could be explored to enhance performance even more.

Specifically, the TransHI training algorithm could be integrated by applying features from other models that have previously sought to improve TransE with commendable results. In particular, TransH [30] is an algorithm that extends TransE by representing each relationship as a hyperplane in a vector space, allowing for a more accurate representation of complex relationships. TransR [23], on the other hand, is another extension of TransE where entities and relationships are represented in separate vector spaces. This distinction allows for a more flexible representation of relationships. Furthermore, TransR employs a transformation matrix that maps entity embeddings to a relationship-specific vector space. In other words, the entity embeddings are adapted and varied based on the specific relationship under consideration. The features of these two algorithms, without affecting the negative triple creation approach, could be integrated into TransHI to achieve a more precise and flexible relationship modeling, potentially yielding even better results than the current version.

Another enhancement that might be beneficial for the initial iteration of TransHI is the implementation of a method to automatically determine the optimal percentage of randomly created negative triples. Since experimental results have highlighted that a certain percentage of random negative triples must be retained and pinpointing the optimal percentage is crucial, instead of experimenting with various values spanning the range from 0 to 100 as done in this work, an automated method to identify this optimal percentage could be implemented.

Lastly, in Chapter 5, it was highlighted that when the preprocessing phase introduces excessive ontological knowledge into the training set, it might be preferable to adopt a method that does not rely on ontology-based triples. It would be beneficial to develop a criterion to quantify the amount of ontological knowledge integrated into the training set. This would assist in determining whether, for the initial training iteration, it would be more appropriate to use TransHI or an alternative approach that does not employ ontology-based triples.



# Bibliography

- [1] K. Ahrabian, A. Feizi, Y. Salehi, W. L. Hamilton, and A. J. Bose. Structure aware negative sampling in knowledge graphs. [abs/2009.11355](#), 2020.
- [2] M. M. Alam, H. Jabeen, M. Ali, K. Mohiuddin, and J. Lehmann. Affinity dependent negative sampling for knowledge graph embeddings. In *DL4KG@ ESWC*, 2020.
- [3] E. Amador-Domínguez, P. Hohenecker, T. Lukasiewicz, D. Manrique, and E. Serrano. An ontology-based deep learning approach for knowledge graph completion with fresh entities. In *International Symposium on Distributed Computing and Artificial Intelligence*, 2019.
- [4] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *Introduction*, page 1–9. Cambridge University Press, 2017.
- [5] F. Bianchi, G. Rossiello, L. Costabello, M. Palmonari, and P. Minervini. Knowledge graph embeddings and explainable AI. [abs/2004.14843](#), 2020.
- [6] P. A. Bonatti, S. Decker, A. Polleres, and V. Presutti. Knowledge Graphs: New Directions for Knowledge Representation on the Semantic Web (Dagstuhl Seminar 18371). *Dagstuhl Reports*, 8(9):29–111, 2019.
- [7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.
- [8] J. Chen, P. Hu, E. Jimenez-Ruiz, O. M. Holter, D. Antonyrajah, and I. Horrocks. Owl2vec\*: Embedding of owl ontologies, 2021.
- [9] Y. Dai, S. Wang, N. N. Xiong, and W. Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5), 2020.
- [10] C. d’Amato, N. F. Quatraro, and N. Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In R. Verborgh, K. Hose, H. Paulheim, P.-A. Champin, M. Maleshkova, O. Corcho, P. Ristoski, and M. Alam, editors, *The Semantic Web*, pages 441–457. Springer International Publishing, 2021.

- [11] S. Dash and A. Gliozzo. Distributional negative sampling for knowledge base completion. 2019.
- [12] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–221, 1993.
- [13] V. Gutiérrez-Basulto and S. Schockaert. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules, 2018.
- [14] W. Hall and K. O’Hara. *Semantic Web*, pages 8084–8104. Springer New York, 2009.
- [15] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Springer, 2021.
- [16] N. Jain, T.-K. Tran, M. H. Gad-Elrab, and D. Stepanova. Improving knowledge graph embeddings with ontological reasoning. In A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, and H. Alani, editors, *The Semantic Web – ISWC 2021*, pages 410–426. Springer International Publishing, 2021.
- [17] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.
- [18] Z. Kaoudi, A. C. M. Lorenzo, and V. Markl. Towards loosely-coupling knowledge graph embeddings and ontology-based reasoning. [abs/2202.03173](https://arxiv.org/abs/2202.03173), 2022.
- [19] M. Kim, J. Cobb, M. J. Harrold, T. Kurc, A. Orso, J. Saltz, A. Post, K. Malhotra, and S. B. Navathe. Efficient regression testing of ontology-driven systems. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, page 320–330. Association for Computing Machinery, 2012.
- [20] B. Kotnis and V. Nastase. Analysis of the impact of negative sampling on link prediction in knowledge graphs, 2018.
- [21] D. Krompaß, S. Baier, and V. Tresp. Type-constrained representation learning in knowledge graphs. In *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I 14*, pages 640–655. Springer, 2015.
- [22] K. Liang, Y. Liu, S. Zhou, W. Tu, Y. Wen, X. Yang, X. Dong, and X. Liu. Knowledge graph contrastive learning based on relation-symmetrical structure. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–12, 2023.



- [23] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), 2015.
- [24] M. Nayyeri, C. Xu, J. Lehmann, and S. Vahdati. Motif learning in knowledge graphs using trajectories of differential equations, 2020.
- [25] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.
- [26] R. Shearer, B. Motik, and I. Horrocks. Hermit: A highly-efficient owl reasoner. In *OWL: Experiences and Directions*, 2008.
- [27] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013.
- [28] Z. Sun, W. Hu, Q. Zhang, and Y. Qu. Bootstrapping entity alignment with knowledge graph embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, page 4396–4402. AAAI Press, 2018.
- [29] G. Töpper, M. Knuth, and H. Sack. Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, page 33–40. Association for Computing Machinery, 2012.
- [30] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI Conference on Artificial Intelligence*, 2014.
- [31] M. Xu. Understanding graph embedding methods and their applications, 2020.
- [32] H. Yu, R. Jiang, B. Zhou, and A. Li. Knowledge-infused pre-trained models for kg completion. In Z. Huang, W. Beek, H. Wang, R. Zhou, and Y. Zhang, editors, *Web Information Systems Engineering – WISE 2020*, pages 273–285. Springer International Publishing, 2020.
- [33] A. Zaveri, D. Kontokostas, S. Hellmann, J. Umbrich, M. Färber, F. Bartscherer, C. Menne, A. Rettinger, A. Zaveri, D. Kontokostas, S. Hellmann, and J. Umbrich. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semant. Web*, 9(1):77–129, 2018.
- [34] W. Zhang, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein, and H. Chen. Iteratively learning embeddings and rules for knowledge graph reasoning. [abs/1903.08948](https://arxiv.org/abs/1903.08948), 2019.
- [35] Z. Zhou, D. Chen, C. Wang, Y. Feng, and C. Chen. Improving knowledge graph embedding via iterative self-semantic knowledge distillation, 2022.



# A | Appendix A

In this section, the list of entailments executed during the "Ontology Axiom Entailment" step (discussed in Sec.4.3.2) is presented. For ease of understanding, the notation used in Tables 2.3, 2.1 and 2.2 is employed to express the axioms. Additionally,  $x, y, z$  represent classes, and  $a, b, c$  represent properties.

$$x\text{-Equiv.Class} \rightarrow y \models y\text{-Equiv.Class} \rightarrow x \quad (\text{A.1})$$

$$a\text{-Equiv.Prop} \rightarrow b \models b\text{-Equiv.Prop} \rightarrow a \quad (\text{A.2})$$

$$x\text{-Disj.Class} \rightarrow y \models y\text{-Disj.Class} \rightarrow x \quad (\text{A.3})$$

$$a\text{-Disj.Prop} \rightarrow b \models b\text{-Disj.Prop} \rightarrow a \quad (\text{A.4})$$

$$a\text{-Inver.Prop} \rightarrow b \models b\text{-Inver.Prop} \rightarrow a \quad (\text{A.5})$$

$$x\text{-Equiv.Class} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-Equiv.Class} \rightarrow z \quad (\text{A.6})$$

$$a\text{-Equiv.Prop} \rightarrow b, b\text{-Equiv.Prop} \rightarrow c \models a\text{-Equiv.Prop} \rightarrow c \quad (\text{A.7})$$

$$x\text{-SubClass} \rightarrow y, y\text{-SubClass} \rightarrow z \models x\text{-SubClass} \rightarrow z \quad (\text{A.8})$$

$$a\text{-SubProp} \rightarrow b, b\text{-SubProp} \rightarrow c \models a\text{-SubProp} \rightarrow c \quad (\text{A.9})$$

$$x\text{-SubClass} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-SubClass} \rightarrow z \quad (\text{A.10})$$

$$a\text{-SubProp} \rightarrow b, b\text{-Equiv.Prop} \rightarrow c \models a\text{-SubProp} \rightarrow c \quad (\text{A.11})$$

$$x\text{-Disj.Class} \rightarrow y, y\text{-Equiv.Class} \rightarrow z \models x\text{-Disj.Class} \rightarrow z \quad (\text{A.12})$$

$$a\text{-Disj.Prop} \rightarrow b, b\text{-Equiv.Prop} \rightarrow c \models a\text{-Disj.Prop} \rightarrow c \quad (\text{A.13})$$

$$a\text{-Inver.Prop} \rightarrow b, b\text{-Equiv.Prop} \rightarrow c \models a\text{-Inver.Prop} \rightarrow c \quad (\text{A.14})$$

$$x\text{-Disj.Class} \rightarrow y, z\text{-SubClass} \rightarrow x \models z\text{-Disj.Class} \rightarrow y \quad (\text{A.15})$$

$$a\text{-Disj.Prop} \rightarrow b, c\text{-SubProp} \rightarrow a \models c\text{-Disj.Prop} \rightarrow b \quad (\text{A.16})$$

$$a\text{-Inver.Prop} \rightarrow b, c\text{-SubProp} \rightarrow a \models c\text{-Inver.Prop} \rightarrow b \quad (\text{A.17})$$

$$a\text{-Domain} \rightarrow x, x\text{-Equiv.Class} \rightarrow y \models a\text{-Domain} \rightarrow y \quad (\text{A.18})$$

$$a\text{-Domain} \rightarrow x, x\text{-SubClass} \rightarrow y \models a\text{-Domain} \rightarrow y \quad (\text{A.19})$$

$$a\text{-Range} \rightarrow x, x\text{-Equiv.Class} \rightarrow y \models a\text{-Range} \rightarrow y \quad (\text{A.20})$$

$$a\text{-Range} \rightarrow x, x\text{-SubClass} \rightarrow y \models a\text{-Range} \rightarrow y \quad (\text{A.21})$$

$$a\text{-type} \rightarrow \text{Irrefl.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Irreflex.} \quad (\text{A.22})$$

$$a\text{-type} \rightarrow \text{Irrefl.}, b\text{-SubProp} \rightarrow a \models b\text{-type} \rightarrow \text{Irreflex.} \quad (\text{A.23})$$

$$a\text{-type} \rightarrow \text{Symm.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Symm.} \quad (\text{A.24})$$

$$a\text{-type} \rightarrow \text{Symm.}, b\text{-SubProp} \rightarrow a \models b\text{-type} \rightarrow \text{Symm.} \quad (\text{A.25})$$

$$a\text{-type} \rightarrow \text{Asymm.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Asymm.} \quad (\text{A.26})$$

$$a\text{-type} \rightarrow \text{Asymm.}, b\text{-SubProp} \rightarrow a \models b\text{-type} \rightarrow \text{Asymm.} \quad (\text{A.27})$$

$$a\text{-type} \rightarrow \text{Transit.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Transit.} \quad (\text{A.28})$$

$$a\text{-type} \rightarrow \text{Transit.}, b\text{-SubProp} \rightarrow a \models b\text{-type} \rightarrow \text{Transit.} \quad (\text{A.29})$$

$$a\text{-type} \rightarrow \text{Funct.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Funct.} \quad (\text{A.30})$$

$$a\text{-type} \rightarrow \text{Inv.Funct.}, a\text{-Equiv.Prop} \rightarrow b \models b\text{-type} \rightarrow \text{Inv.Funct.} \quad (\text{A.31})$$

# A | Appendix B

In this appendix, tables containing the experimental results are provided, from which the best outcomes were extracted for the composition of the tables in Chapter 5.

Specifically, these results pertain to the initial iteration of the framework, where the primary objective is to determine the optimal percentage of random negative triples to incorporate. This percentage has been ascertained for all three variants of TransHI presented in Chapter 5, namely "TransHI- ontology-based only", "TransHI- structure-based only" and the comprehensive TransHI.

To pinpoint this percentage, trainings were conducted using percentages ranging from 10 to 100, in increments of 10. The subsequent tables thus capture these multiple attempts.

The attempt yielding the best H@10/MR metric corresponds to the data presented in the tables of Chapter 5.

For reference, the performances of TransE and TransOWL, both applied to original and pre-processed data, are also included in each table.

	NotTypeOf		TypeOf		General			%Ontology-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	587,07	32,46	692,29	9,75	1279,36	42,21	0,0329931	-
TRANS- OWL (original data)	621,06	32,24	493,46	13,20	1114,52	45,44	0,0407709	-
TRANS-E (preprocessed data)	611,63	33,29	295,13	12,78	906,76	46,07	0,0508073	-
TRANS- OWL (preprocessed data)	641,76	33,72	287,89	12,71	929,65	46,43	0,0499435	-
TransHI-ontology based only	604,99	33,37	296,08	12,71	901,07	46,08	0,0511392	10
	606,72	33,30	296,64	12,64	903,36	45,94	0,0508546	20
	612,81	33,51	294,67	12,54	907,48	46,05	0,0507449	30
	602,90	33,30	293,72	12,65	896,62	45,95	0,0512480	40
	588,23	33,37	294,87	12,66	883,10	46,03	0,0521232	50
	608,49	33,39	293,19	12,75	901,68	46,14	0,0511711	60
	602,59	32,83	295,86	12,67	898,45	45,50	0,0506428	70
	613,49	33,21	294,00	12,77	907,49	45,98	0,0506672	80
	587,16	32,64	294,82	12,70	881,98	45,34	0,0514071	90
	593,58	32,11	294,43	12,76	888,01	44,87	0,0505287	100

Figure A.1: DBPEDIA15K: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Structure-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	587,07	32,46	692,29	9,75	1279,36	42,21	0,0329931	-
TRANS- OWL (original data)	621,06	32,24	493,46	13,20	1114,52	45,44	0,0407709	-
TRANS-E (preprocessed data)	611,63	33,29	295,13	12,78	906,76	46,07	0,0508073	-
TRANS- OWL (preprocessed data)	641,76	33,72	287,89	12,71	929,65	46,43	0,0499435	-
TransHI-structure based only	577,79	32,81	284,25	12,92	862,04	45,73	0,0530486	10
	602,69	33,32	283,89	12,89	886,58	46,21	0,0521216	20
	616,61	33,35	283,18	12,88	899,79	46,23	0,0513787	30
	581,74	33,41	284,46	12,79	866,20	46,20	0,0533364	40
	601,33	33,29	284,06	12,95	885,39	46,24	0,0522256	50
	616,92	32,70	284,51	12,84	901,43	45,54	0,0505197	60
	619,00	33,13	282,83	12,81	901,83	45,94	0,0509409	70
	612,17	32,87	285,31	12,86	897,48	45,73	0,0509538	80
	634,03	32,44	284,74	12,90	918,77	45,34	0,0493486	90
	712,15	31,15	282,88	12,89	995,03	44,04	0,0442600	100

Figure A.2: DBPEDIA15K: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Ontology/Structure based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	587,07	32,46	692,29	9,75	1279,36	42,21	0,0329931	-
TRANS- OWL (original data)	621,06	32,24	493,46	13,20	1114,52	45,44	0,0407709	-
TRANS-E (preprocessed data)	611,63	33,29	295,13	12,78	906,76	46,07	0,0508073	-
TRANS- OWL (preprocessed data)	641,76	33,72	287,89	12,71	929,65	46,43	0,0499435	-
TransHI	604,23	33,77	284,31	12,84	888,54	46,61	0,0524568	10
	603,89	32,97	283,28	12,73	887,17	45,70	0,0515121	20
	577,47	33,15	284,33	12,89	861,80	46,04	0,0534231	30
	601,93	32,68	284,10	12,92	886,03	45,60	0,0514655	40
	597,57	32,89	283,01	12,81	880,58	45,70	0,0518976	50
	582,59	32,40	282,94	12,96	865,53	45,36	0,0524072	60
	605,17	32,45	283,58	12,90	888,75	45,35	0,0510267	70
	605,14	31,97	283,27	12,90	888,41	44,87	0,0505060	80
	614,38	31,11	284,94	13,01	899,32	44,12	0,0490593	90
	757,30	26,58	285,24	12,98	1042,54	39,56	0,0379458	100

Figure A.3: DBPEDIA15K: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Ontology-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7417,08	19,24	587,19	8,71	8004,27	27,95	0,0034919	-
TRANS- OWL (original data)	7455,49	19,21	580,29	8,68	8035,78	27,89	0,0034707	-
TRANS-E (preprocessed data)	7437,56	19,68	466,28	9,13	7903,84	28,81	0,0036451	-
TRANS- OWL (preprocessed data)	7292,21	19,84	473,94	9,04	7766,15	28,88	0,0037187	-
TransHI-ontology based only	7343,10	19,68	470,36	9,19	7813,46	28,87	0,0036949	10
	7356,42	19,57	475,74	9,12	7832,16	28,69	0,0036631	20
	7442,58	19,59	468,85	9,06	7911,43	28,65	0,0036213	30
	7495,70	19,49	468,48	9,12	7964,18	28,61	0,0035923	40
	7447,02	19,65	470,98	9,19	7918,00	28,84	0,0036423	50
	7498,53	19,83	472,81	9,25	7971,34	29,08	0,0036481	60
	7486,44	19,19	474,71	9,17	7961,15	28,36	0,0035623	70
	7491,33	19,41	477,29	9,27	7968,62	28,68	0,0035991	80
	7455,18	19,76	470,93	9,27	7926,11	29,03	0,0036626	90
	7519,87	19,30	478,44	9,33	7998,31	28,63	0,0035795	100

Figure A.4: YAGO: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Structure-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7417,08	19,24	587,19	8,71	8004,27	27,95	0,0034919	-
TRANS- OWL (original data)	7455,49	19,21	580,29	8,68	8035,78	27,89	0,0034707	-
TRANS-E (preprocessed data)	7437,56	19,68	466,28	9,13	7903,84	28,81	0,0036451	-
TRANS- OWL (preprocessed data)	7292,21	19,84	473,94	9,04	7766,15	28,88	0,0037187	-
TransHI-structure based only	7281,56	19,53	463,85	9,21	7745,41	28,74	0,0037106	10
	7312,94	19,95	462,75	9,11	7775,69	29,06	0,0037373	20
	7337,33	19,83	469,23	9,12	7806,56	28,95	0,0037084	30
	7100,72	20,19	465,14	9,18	7565,86	29,37	0,0038819	40
	7166,85	19,89	471,59	9,16	7638,44	29,05	0,0038031	50
	7013,95	20,15	473,77	9,25	7487,72	29,40	0,0039264	60
	7108,65	20,21	480,42	9,19	7589,07	29,40	0,0038740	70
	7109,02	20,26	485,55	9,22	7594,57	29,48	0,0038817	80
	7193,77	20,37	502,66	9,18	7696,43	29,55	0,0038394	90
	7391,55	19,91	521,88	9,09	7913,43	29,00	0,0036647	100

Figure A.5: YAGO: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Ontology/Strucure based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7417,08	19,24	587,19	8,71	8004,27	27,95	0,0034919	-
TRANS- OWL (original data)	7455,49	19,21	580,29	8,68	8035,78	27,89	0,0034707	-
TRANS-E (preprocessed data)	7437,56	19,68	466,28	9,13	7903,84	28,81	0,0036451	-
TRANS- OWL (preprocessed data)	7292,21	19,84	473,94	9,04	7766,15	28,88	0,0037187	-
TransHI	7343,84	19,87	466,10	9,12	7809,94	28,99	0,0037119	10
	7287,10	19,97	467,74	9,19	7754,84	29,16	0,0037602	20
	7165,61	20,39	463,26	9,16	7628,87	29,55	0,0038734	30
	7317,79	19,81	469,41	8,98	7787,20	28,79	0,0036971	40
	7183,08	20,19	474,55	9,20	7657,63	29,39	0,0038380	50
	7020,42	20,31	472,94	9,15	7493,36	29,46	0,0039315	60
	7330,67	17,36	516,97	9,21	7847,64	26,57	0,0033857	70
	7270,85	20,04	483,38	9,19	7754,23	29,23	0,0037696	80
	7071,91	20,34	492,65	9,14	7564,56	29,48	0,0038971	90
	7486,08	20,09	510,51	9,07	7996,59	29,16	0,0036466	100

Figure A.6: YAGO: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Ontology-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7162,08	19,01	2872,45	6,55	10034,53	25,56	0,0025472	-
TRANS- OWL (original data)	9622,40	15,54	2263,09	6,52	11885,49	22,06	0,0018560	-
TRANS-E (preprocessed data)	6112,15	19,79	3527,43	6,12	9639,58	25,91	0,0026879	-
TRANS- OWL (preprocessed data)	5730,69	19,89	4048,52	3,89	9779,21	23,78	0,0024317	-
TransHI-ontology based only	6589,78	19,61	3748,56	5,58	10338,34	25,19	0,0024366	10
	6150,79	19,70	3672,64	6,55	9823,43	26,25	0,0026722	20
	6437,51	20,01	3617,83	6,60	10055,34	26,61	0,0026464	30
	6465,94	19,90	3605,93	6,37	10071,87	26,27	0,0026083	40
	6327,32	19,77	3460,93	6,38	9788,25	26,15	0,0026716	50
	6331,12	19,62	3557,94	7,30	9889,06	26,92	0,0027222	60
	6201,27	19,66	3525,23	6,58	9726,50	26,24	0,0026978	70
	6193,88	19,79	3524,10	6,07	9717,98	25,86	0,0026610	80
	6026,96	19,61	3525,05	6,49	9552,01	26,10	0,0027324	90
	6697,95	19,84	3489,73	6,30	10187,68	26,14	0,0025658	100

Figure A.7: NELL: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Structure-based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7162,08	19,01	2872,45	6,55	10034,53	25,56	0,0025472	-
TRANS- OWL (original data)	9622,40	15,54	2263,09	6,52	11885,49	22,06	0,0018560	-
TRANS-E (preprocessed data)	6112,15	19,79	3527,43	6,12	9639,58	25,91	0,0026879	-
TRANS- OWL (preprocessed data)	5730,69	19,89	4048,52	3,89	9779,21	23,78	0,0024317	-
TransHI-structure based only	5824,20	20,04	2561,94	8,13	8386,14	28,17	0,0033591	10
	5934,96	20,14	2441,94	8,63	8376,90	28,77	0,0034344	20
	5716,35	20,04	2507,50	8,29	8223,85	28,33	0,0034449	30
	5780,65	20,03	2557,61	8,00	8338,26	28,03	0,0033616	40
	5592,37	19,78	2551,70	8,08	8144,07	27,86	0,0034209	50
	5370,85	19,73	2622,63	8,56	7993,48	28,29	0,0035391	60
	5943,34	19,84	2551,38	8,70	8494,72	28,54	0,0033597	70
	5579,57	19,79	2593,26	8,33	8172,83	28,12	0,0034407	80
	5617,28	19,76	2522,67	7,83	8139,95	27,59	0,0033895	90
	6710,06	19,50	2610,23	7,23	9320,29	26,73	0,0028679	100

Figure A.8: NELL: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red).

	NotTypeOf		TypeOf		General			%Ontology/Structure based negatives used
	MR	H@10	MR	H@10	MR	H@10	H@10/MR	
TRANS-E (original data)	7162,08	19,01	2872,45	6,55	10034,53	25,56	0,0025472	-
TRANS- OWL (original data)	9622,40	15,54	2263,09	6,52	11885,49	22,06	0,0018560	-
TRANS-E (preprocessed data)	6112,15	19,79	3527,43	6,12	9639,58	25,91	0,0026879	-
TRANS- OWL (preprocessed data)	5730,69	19,89	4048,52	3,89	9779,21	23,78	0,0024317	-
TransHI	5803,93	20,14	2597,72	7,31	8401,65	27,45	0,0032672	10
	5886,94	19,82	2596,29	7,26	8483,23	27,08	0,0031922	20
	5855,15	19,81	2504,53	7,87	8359,68	27,68	0,0033111	30
	5700,53	20,04	2613,31	7,47	8313,84	27,51	0,0033089	40
	5380,67	19,80	2557,24	8,05	7937,91	27,85	0,0035085	50
	5776,84	19,53	2565,50	7,85	8342,34	27,38	0,0032821	60
	5905,00	19,88	2582,74	7,57	8487,74	27,45	0,0032341	70
	5947,67	19,32	2648,60	7,50	8596,27	26,82	0,0031200	80
	6024,45	19,48	2690,50	7,69	8714,95	27,17	0,0031176	90
	6676,78	18,78	2749,90	7,68	9426,68	26,46	0,0028069	100

Figure A.9: NELL: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red).



## List of Figures

2.1	Representation of a graph and its non-oriented version. . . . .	6
2.2	CC of node $v_i$ in a non-oriented graph (left) and an oriented graph (right). . . .	8
2.3	Example of adjacency matrix and adjacency list of an oriented graph. . . . .	10
2.4	Generic triple (h,r,t) in a knowledge graph . . . . .	11
2.5	Example of a simple knowledge graph. . . . .	11
2.6	Semantic schema graph example. . . . .	12
2.7	A simple KG with 2 classes and 3 instances. . . . .	13
2.8	Example of a simple ontology. . . . .	17
2.9	Example of 1-hot encoding technique. . . . .	26
2.10	Embedding of a knowledge graph for a machine learning task. . . . .	28
3.1	Graphical representation of the SANS approach. . . . .	37
3.2	Pipeline of iterative ontology-based reasoning. . . . .	38
4.1	Schema of the TransHI approach. . . . .	43
4.2	Schema of the preprocessing phase of TransHI. . . . .	45
4.3	Consistency check representation. . . . .	46
4.4	Binary search for consistent subgroups representation. . . . .	47
4.5	Example of the problematic related to the binary search method. . . . .	50
4.6	Example of the method to collect the evidence supporting class membership. . .	52
4.7	Examples of ontological axioms entailments. . . . .	56
4.8	Schema of the TransHI component for the first iteration training. . . . .	59
4.9	Example of application of the SANS method on a positive triple. . . . .	69
4.10	Selection of neighbors with low diversification. . . . .	69
4.11	Selection of neighbors with high diversification. . . . .	70
4.12	Schema of the "Training Data Update" phase of TransHI. . . . .	71
4.13	Schema of the TransHI component for the next iterations training. . . . .	75
5.1	DBPEDIA15K: TransHI's first iteration results (better = green, worse = red). .	90
5.2	YAGO: TransHI's first iteration results (better = green, worse = red). . . . .	91
5.3	NELL: TransHI's first iteration results (better = green, worse = red). . . . .	92
5.4	DBPEDIA15K: TransHI's first 10 iterations results (better = green, worse = red). .	95

5.5	YAGO: TransHI's first 10 iterations results (better = green, worse = red). . . . .	95
5.6	NELL: TransHI's first 10 iterations results (better = green, worse = red). . . . .	96
A.1	DBPEDIA15K: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	107
A.2	DBPEDIA15K: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	108
A.3	DBPEDIA15K: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	108
A.4	YAGO: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	108
A.5	YAGO: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	109
A.6	YAGO: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	109
A.7	NELL: TransHI using only ontology-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	109
A.8	NELL: TransHI using only structure-based negatives. Results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	110
A.9	NELL: TransHI's results of the first iteration with different percentages of random negatives (better = green, worse = red). . . . .	110

# List of Tables

2.1	Main OWL axioms for individuals . . . . .	19
2.2	Main OWL axioms for properties . . . . .	20
2.3	Main OWL axioms for classes . . . . .	21
4.1	Types of negative triples generable using ontology axioms. . . . .	64
5.1	Features of the KGs used for TransHI's evaluation. . . . .	86
5.2	Results of the "Inconsistencies Correction" step. . . . .	86
5.3	Axioms contained in and entailed from the three ontologies. . . . .	87
5.4	Results of the "Positive Triples Augmentation" step. . . . .	88
5.5	Percentages of entities without an associated class. . . . .	90

