Modelling Academic Data with Knowledge Graphs

Marc Parcerisa Conesa Robert Vila Alsina
Facultat d'Informàtica de Barcelona (FIB), UPC, 08034 Barcelona, Spain

***Abstract*—Knowledge graphs enable structured, semantically rich representations of complex data. This project explores RDF-based knowledge graphs for modelling academic research. In the first approach DBpedia data has been analised using SPARQL in GraphDB. Then, we constructed a custom ontology with a TBox and ABox using data from Semantic Scholar, resulting in over 625,000 triples across 15 classes and 30 properties. Key queries demonstrate applications such as detecting peer review conflicts and identifying top authors by topic. Finally, we generate knowledge graph embeddings (KGEs) using TransE, ComplEx, and RotatE, selecting RotatE as the best performer. These embeddings are then used to recommend potential paper reviewers.**

I. Introduction

In the first project, we used Property Graphs to model academic research data. Property graphs offer a simple and efficient way to navigate and perform queries on graph data. However, a richer representation with semantic reasoning capabilities is sometimes needed for integration and knowledge inference. For this reason, and to get hands-on with the second half of the course content, **knowledge graphs** will be introduced in this second assignment.

A knowledge graph is a graph where every node is represented with a unique identifier and can be universally referred to. However, data is solely represented as nodes and edges, without special constructs such as labels or attributes, which means that data integration is specially tricky.

In this project, we will use a GraphDB database under the RDF entailment for storing, integrating, querying and deriving information about several *Semantic Scholar* publications, their authors, and their publication venues. All queries throughout this report were written in the SPARQL language.

This report is organized as follows: In section A (not included in the report), we explored a knowledge graph we extracted from DBpedia[1] using SPARQL and the tools provided in GraphDB. Section II describes the steps we followed to create an ontology from the *Semantic Scholar* graph database, using RDF semantics to define a TBOX that strictly models all the information available, as well as an ABOX with the real data[2]. Some queries are executed to show to power of the ontology. Finally, in Section III we define Knowledge Graph Embeddings (KGEs) for Machine Learning purposes.

All the code explained throughout this report is available attached, or in the public GitHub repository AimbotParce/MDS-SDM-KnowledgeGraphs. The attached files are missing important pieces of the code that are located inside common library directories, so attached code might not work.

[1]DBpedia provides an open-source solution that extracts a knowledge graph from the information created in the Wikipedia project. https://www.dbpedia.org/.

[2]Strictly speaking, Knowledge graphs in the entailment realm of RDF do not distinguish between TBOX and ABOX, although the graph from this project was made to allow for a clean separation of them

II. Ontology Creation

A knowledge graph can be thought of as a conjunction of an ontology, also called a Terminological Box (TBox), and a set of data, also called an Assertional Box (ABox). The RDFS syntax provides a small set of predicates that are flexible enough to represent a vast amount of relationships in the TBox, whilst also being rich enough to allow for definitions of strict conditions.

This project aims at modelling the relations that authors, papers, journals, conferences and workshops, and reviews have in the context of the representation of scientific papers and their citations. To do so, the library *RDFLib* for *Python* provides a basic set of tools for querying and manipulating knowledge graphs either in memory or hosted in Knowledge Databases such as *GraphDB*.

A. TBox definition

In this section, we used RDFLib in Python to create a TBOX based on the text that explains the relations between papers, authors, journals, etc. Although the paper is very similar to the one modelled in the first project, there are some key changes. Besides the content of the text, we have made some assumptions to enrich the content of the network and make it more realistic:

- The **Review** written by the *Authors* about a *Paper* has a *Verdict* and a *Content*.
- Apart from the *Abstract*, a **Paper** node also contains the *Title* and its whole *Content*.

Based on the information retrieved from the explanation in the statement and the assumptions made to enrich the graph, we also took important decisions about the architecture of the nodes and edges:

- **Reviews** are modelled as a single node and not as a relation between authors and papers. This is to be able to add the *Verdict* and *Content* explained in the assumptions.
- **Topics** are modelled as a single node with *Literal* properties referring to their keywords, to allow queries relating the papers about the same topic.
- A **Paper** is published in a **Publication Venue**, which can be either a **Journal Volume** or the **Proceedings** of a **Scientific Forum**, that can be either a **Workshop** or a **Conference**. Finally, all **Journal Volumes** are volumes of some **Journal**:

$$JournalVolume, Proceedings \sqsubseteq PublicationVenue$$

$$PublicationVenue \sqsubseteq JournalVolume \cup Proceedings$$

$$JournalVolume \cap Proceedings \sqsubseteq \bot$$

$$\exists isProceedingsOf \sqsubseteq Proceedings$$

$$\exists isProceedingsOf^{-} \sqsubseteq ScientificForum$$

$$Conference, Workshop \sqsubseteq ScientificForum$$

$$ScientificForum \sqsubseteq Conference \cup Workshop$$

$$Conference \cap Workshop \sqsubseteq \bot$$

- An **Author** is that which **writes** a paper, and it can do so as its **corresponding author**. In other words, all who are corresponding authors of a paper have also written it:

$$isCorrespondingAuthor \sqsubseteq writesPaper$$

If instead of limiting to RDF, we decided to use a more richer syntax, such as OWL, some stricter restrictions could be added such as the following:

- One *Paper* is only *published in* one *Publication Venue*.
- A *Proceedings* venue can be only *proceedings of* one *Scientific Forum*.

The complete schema of the defined TBox can be seen in figure 1. Note that no triples of the form `?s rdf:type ?p` is present, as all properties of the kind *domain*, *range*, *subClassOf* and *subPropertyOf* already entail the type of their constituents.

B. ABOX definition

The following steps were performed programmatically to obtain, transform and load data from the Semantic Scholar database into ours, having a freshly defined terminology (TBox).

1) Script `download_graph.py` uses the Semantic Scholar API to:
   a) Perform a search over the papers on their database based on a user-provided query (in our example, "machine learning"), ordering them by descending citation count, with a limit of 5000 elements.
   b) Given the set of papers, retrieve their details, such as authors, abstract, AI-generated TLDR, etc.
   c) Write the papers and details to disk in JSONL format.
   d) For each paper, query again the database to retrieve a list of its references, which contain a *cited paper*, a *citing paper* (the one being queried for), as well as some details about the reference that are ignored in this project.
   e) Write the references to disk.

   In the end, 5,000 papers were retrieved, along with more than 230,000 references.
2) Script `load_abox.py` uses multi-threading to, in parallel:
   a) Load sequentially all the information about the papers.
   b) For each paper, ensure it has a proper paperID (given by the Semantic Scholar API).
   c) Add the triples for the paper's *title*, *abstract* as reported by the API.
   d) If a TLDR is available, add it as the paper's *content*.
   e) For each of the paper's *fields of study*, add the triples relating the paper to it, as well as the triples relating it to a *keyword*. Note: we didn't have access to topics and keywords, so instead a topic is a node whose URI is a field of study in the form of a normalized text[3], and it has a Literal property which is the field of study without normalizing.
   f) Depending on the type (if present) of the *publication venue* of the paper, add the proper triples, following the TBox explained before. If the type of the publication venue is not present, the generic *isPublishedIn* property and *PublicationVenue* class are used.
      Note: Because we don't have the edition of the proceedings present in the Semantic Scholar data, we assume that the year of publication is the same as the edition of the Forum, or in other words, that there's only one edition of the Forum per year.
   g) Insert also the triples relating the publication venue to its *Journal*, or its *Conference* or *Workshop*.
   h) For each of the paper's authors, provided that they have a proper ID defined, add the triples that relate them as a writer (or corresponding author, if they are the first in the list), as well as the triple defining the author's name.
   i) Finally, load sequentially all the information about the references, and add the *paperCites* triples.
3) Script `generate_reviews.py` also uses multi-threading to:
   a) Query the URIs of all papers in the GraphDB database.
   b) For each paper, use the following query to generate a list of reviewer candidates:

```
SELECT DISTINCT ?reviewer WHERE {
```

[3]Lowercase, without spaces or slashes.

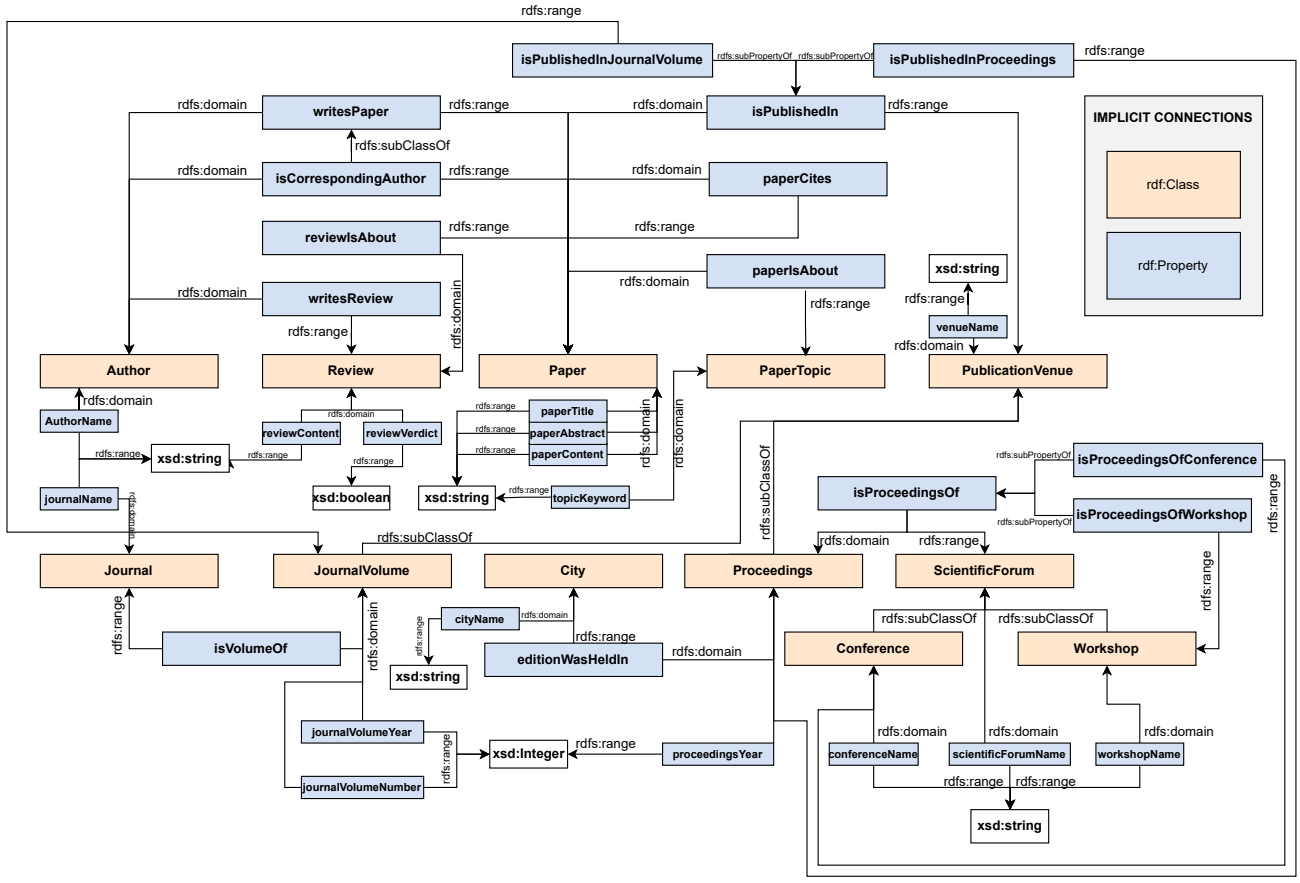

Fig. 1: Complete TBox schema defined to represent relations between papers, their citations, their authors and the venues they were published in. For readability, the `rdf:Class` and `rdf:Property` connections are implicit. Moreover, the `xsd:string` appears more than one time but they all point to the same node.

```
    ?reviewer P:writesPaper ?paper2 .
    ?paper2 a P:Paper ;
        P:paperIsAbout ?topic .
    <{paper}> a P:Paper ;
        P:paperIsAbout ?topic .
    ?topic a P:PaperTopic .
    FILTER NOT EXISTS {
        ?reviewer P:writesPaper <{paper}>
    }
} LIMIT 100
```

Note that this query retrieves at most 100 authors that have written some publication of a topic related to a given paper, provided that they haven't also written the paper itself.

c) Choose randomly between 3 and 8 reviewers from the list (if available), and add the triples defining the new review, its verdict, its content (a dummy string, for now), and the triples they relate it to the author and the paper.

Note that all scripts have a toggleable parameter to control whether *extra* assertions are added to the graph of the form `?s rdf:type ?o`, which aren't needed given that all of the properties used have a unique range and domain (Our TBox is *complete*[4]).

To have an idea of the magnitude of the graph we are working with, it is important to retrieve statistics about the classes and properties. In first place, the **number of classes** is 15, the **number of properties** is 30 and the **total number of triples** is 625,915. Moreover, in Table I we counted the number of instances for every one of the 15 classes and in Table II the same for the properties.

C. Querying the ontology

1) Reviewer–Author Past Collaboration Conflicts: A review should be an objective verdict only on the content of the reviewed paper. Because reviewers are also authors of other papers, it may be the case that there is a conflict of interest where the reviewer has previously worked with a reviewed author, creating a bias. The reason for the creation of the first query is precisely to detect these cases, in which we sort these situations in descending order of the number of collaborations between the reviewer and the author of a paper.

[4]It contains all relevant general knowledge about the domain.

TABLE I: Number of Instances per Class in the Research Publications Knowledge Graph

Class	Instance Count
rdf:Property	37
rdfs:Class	28
rdf:List	1
rdfs:Datatype	3
rdfs:ContainerMembershipProperty	1
p:Paper	140,688
p:PaperTopic	18
p:Journal	719
p:JournalVolume	1,496
p:PublicationVenue	2,249
p:Conference	259
p:ScientificForum	259
p:Proceedings	675
p:Author	20,495
p:Review	30,397

```
PREFIX : http://localhost:7200/academia-sdm
SELECT ?reviewerName ?authorName ?paper
    (COUNT(DISTINCT ?pastPaper)
    AS ?coauthoredCount)
WHERE {
  ?reviewer a :Author ;
    :writesReview ?review ;
    :authorName ?reviewerName .
  ?review a :Review ;
      :reviewIsAbout ?paper .
  ?paper a :Paper .
  ?author a :Author;
    :writesPaper ?paper ;
    :authorName ?authorName .
  FILTER(?reviewer != ?author)
#Find past papers co-authored by both
  ?reviewer :writesPaper ?pastPaper .
  ?author :writesPaper ?pastPaper .
  FILTER(?pastPaper != ?paper)
}
GROUP BY ?reviewerName ?authorName ?paper
ORDER BY DESC(?coauthoredCount)
```

2) Top Authors per Topic: The objective of this query is to find the most important authors for each topic. The performance of a scientist based on published papers can be measured in several ways: number of published papers, impact factor, h-index, or number of citations. In this case, we focus on citation count as a proxy for impact.

Due to SPARQL's limited support for ranking and filtering within grouped results, the query is split into two subqueries: one to compute the maximum citation count per topic, and another to retrieve the corresponding author. The query starts by counting the number of citations of all the papers an author has written related to a specific topic. It next saves the maximum number of citations per topic and compares it with the second subquery, in which it fetches the winning author.

```
PREFIX : http://localhost:7200/academia-sdm
SELECT ?topic ?maxCitations ?otherAuthor
WHERE {
{
    SELECT ?topic (MAX(?citations)
        AS ?maxCitations)
    WHERE {
        SELECT ?author ?topic
            (COUNT(?paper2) AS ?citations)
        WHERE {
            ?author a :Author;
                :writesPaper ?paper .
            ?paper a :Paper ;
                :paperIsAbout ?topic .
            ?topic a :PaperTopic .
            ?paper2 a :Paper ;
                :paperCites ?paper .
            ?author2 a :Author;
                :writesPaper ?paper2 .
            FILTER( ?paper2 != ?paper )
            FILTER( ?author != ?author2 )
        }
        GROUP BY ?author ?topic
        ORDER BY DESC(?citations)
    }
    GROUP BY ?topic
    ORDER BY DESC(?maxCitations)
}
{
SELECT ?otherAuthor ?otherTopic
(COUNT(?otherPaper2) AS ?otherCitations)
WHERE {
    ?otherAuthor a :Author;
        :writesPaper ?otherPaper .
    ?otherPaper a :Paper ;
        :paperIsAbout ?otherTopic .
    ?otherTopic a :PaperTopic .
    ?otherPaper2 a :Paper ;
        :paperCites ?otherPaper .
    ?otherAuthor2 a :Author;
        :writesPaper ?otherPaper2 .
  FILTER( ?otherPaper2 != ?otherPaper )
  FILTER( ?otherAuthor != ?otherAuthor2 )
    }
    GROUP BY ?otherAuthor ?otherTopic
    ORDER BY DESC(?otherCitations)
}
FILTER ( ?otherTopic = ?topic )
FILTER ( ?maxCitations = ?otherCitations )
}
```

III. Graph Embeddings

Having defined a detailed ontology and instantiated a rich academic knowledge graph with real-world data, we now move beyond symbolic reasoning and querying capabilities toward embedding-based learning methods. Knowledge Graph Embeddings (KGEs) allow us to encode entities and relations from the graph into continuous vector spaces, enabling the application of machine learning techniques for downstream tasks such as link prediction, clustering, and classification. In this section, we leverage *PyKEEN* to generate these embeddings from our constructed RDF graph, systematically analyze different embedding models, and assess their suitability for capturing the underlying semantics of academic data.

TABLE II: Triple Count per Property in the ABOX

Property	Count	Property	Count
p:paperCites	220,365	rdf:type	197,325
p:reviewContent	30,397	p:reviewIsAbout	30,397
p:writesReview	30,397	p:reviewVerdict	30,397
p:writesPaper	25,690	p:authorName	20,495
p:paperIsAbout	7,250	p:paperAbstract	5,000
p:paperTitle	5,000	p:isCorrespondingAuthor	4,971
p:paperContent	4,867	p:isPublishedIn	3,706
p:isPublishedInJournalVolume	2,194	p:isVolumeOf	1,496
p:journalVolumeNumber	1,269	p:isPublishedInProceedings	1,183
p:journalName	719	p:isProceedingsOfConference	675
p:isProceedingsOf	675	p:proceedingsYear	675
p:conferenceName	259	p:scientificForumName	259
p:venueName	78	rdfs:subPropertyOf	46
rdfs:subClassOf	40	rdfs:domain	36
rdfs:range	36	p:topicKeyword	18

A. Importing the data

To prepare the knowledge graph for embedding generation, we first exported the full set of triples from GraphDB in TSV format using SPARQL. Although we initially considered removing TBox-related triples to simplify the structure and focus the embeddings on instance-level data, we ultimately decided to retain all nodes, including those related to the ontology, because SPARQL inference was enabled and all `rdf:type` assertions were already materialized in the results of the queries. This ensured that class-level information could still contribute to the embedding process. However, we excluded all literals (for example *authorName*, *journalName* and *paperTitle*) from the dataset, as they do not participate in meaningful relational structure and are not suitable for embedding within the PyKEEN framework.

B. Getting familiar with KGEs

We begin our exploration of knowledge graph embeddings by training a simple yet foundational model: *TransE*. As one of the earliest and most interpretable KGE models, TransE represents relationships as translations in a vector space. Using the exported data of our knowledge graph, we trained *TransE* with a basic configuration to generate embeddings for all entities and relations. With these embeddings, we then explored how vector arithmetic can be used to retrieve semantically related nodes:

1) We begin by randomly choosing a paper from the dataset, ensuring that it has at least one reference:
```
SELECT ?p WHERE {
    ?p a :Paper ;
       :paperCites ?c .
} ORDER BY RAND() LIMIT 1
```
2) The way *TransE* is trained, instead of predicting an embedding, it predicts a score for any given triple based on its embeddings (similar to the SkipGram training strategy used for Word2Vec embeddings). Thus, we cannot use the model to actually predict a possible embedding. We must either rely on computing the score for all possible tails (all nodes in the graph) and get the highest one (this way the result is surely a node inside the graph), or mathematically compute the *translation*:
```
p = "http://localhost:7200/academia-sdm#"
paper = f"<{p}{paper_uri}>"
cites = f"<{p}paperCites>"
entt_emb = model.entity_representations[0]
rel_emb = model.relation_representations[0]
sub_id = training.entity_to_id[paper]
sub_rep = entt_emb(indices=[sub_id])
rel_id = training.relation_to_id[cites]
rel_rep = rel_emb(indices=[rel_id])
pred_paper = sub_rep + rel_rep
```
We could then check the distance[5] between the resulting vector and the embeddings of all possible *tails* to choose the most probable.
3) Having the most likely paper embedding, we could compute the most likely author embedding by performing the same steps, this time adding the *writesPaper* relation:
```
writes = f"<{p}writesPaper>"
write_id = training.relation_to_id[writes]
write_rep = rel_emb(indices=[write_id])
pred_auth = pred_paper + write_rep
```
4) Finally, we can obtain the most likely author from our knowledge graph in terms of euclidean distance:
```
b_auth = None
b_dist = -1.0
for aid, ind in entt_to_id.items():
    rep = entt_emb(indices=[ind])
    dist = torch.cdist(pred_auth, rep)
    if dist < b_dist or b_auth is None:
        b_auth = aid
        b_dist = dist
```
What we find is that the result of this operation is not necessarily an author, so we have to limit solutions to authors. We do so by first obtaining all nodes of type *Author*:
```
select * where {
    ?s a :Author .
}
```

[5]We should use the distance function with which the model was trained originally (typically, either the L1 norm or the euclidian distance).

And downloading them in TSV format, to add a filter to the script.

In the end, the author that is most likely to have written a paper cited by paper "Boosting methods for multi-class imbalanced data classification: an experimental review" is "Haitao Luo", author of the paper "KOBAS-i: intelligent prioritization and exploratory visualization of biological functions for gene enrichment analysis", a paper which is NOT cited by the first one.

1) Limitations of TransE Embeddings: Because *TrasnE* is a transactional embedding model that models relations of the form

$$\mathbf{head} + \mathbf{rel} \approx \mathbf{tail},$$

from the triples

$$(Author1, writes, Paper1)$$

$$(Author2, writes, Paper1)$$

$$(Author1, writes, Paper2),$$

would derive the relations

$$\mathbf{Author1} + \mathbf{writes} \approx \mathbf{Paper1}$$

$$\mathbf{Author2} + \mathbf{writes} \approx \mathbf{Paper1}$$

$$\mathbf{Author1} + \mathbf{writes} \approx \mathbf{Paper2}.$$

These relations imply that $\mathbf{Author1} \approx \mathbf{Author2}$ and $\mathbf{Paper1} \approx \mathbf{Paper2}$. In other words, relations with cardinality higher than 1 are poorly represented in *TransE*.

To circumvent this inconvenience, models such as *TransH* were developed. To do so, *TransH* uses a per-relation representation of the entities, created by projecting the "global" entity representations onto a relation-specific hyperplane. Thus, instead of having a single representation for each entity and one for each relation, this model learns, for relations, both a normal vector to a hyperplane, and a translation within that hyperplane:

$$\mathbf{head}_\perp = \mathbf{head} - \mathbf{w_r}^\top \cdot \mathbf{head} \cdot \mathbf{w_r}$$

$$\mathbf{tail}_\perp = \mathbf{tail} - \mathbf{w_r}^\top \cdot \mathbf{tail} \cdot \mathbf{w_r}$$

$$\mathbf{head}_\perp + \mathbf{r} \approx \mathbf{tail}_\perp.$$

Similarly to high cardinality relations, *TransE* also struggles with symmetric relations. Suppose we have the triples

$$(Author1, collaboratesWith, Author2)$$

$$(Author2, collaboratesWith, Author1).$$

Then, by the translation relationship shown before, *TransE* would derive the relations

$$\mathbf{Auhtor1} + \mathbf{collaboratesWith} \approx \mathbf{Author2}$$

$$\mathbf{Auhtor2} + \mathbf{collaboratesWith} \approx \mathbf{Author1},$$

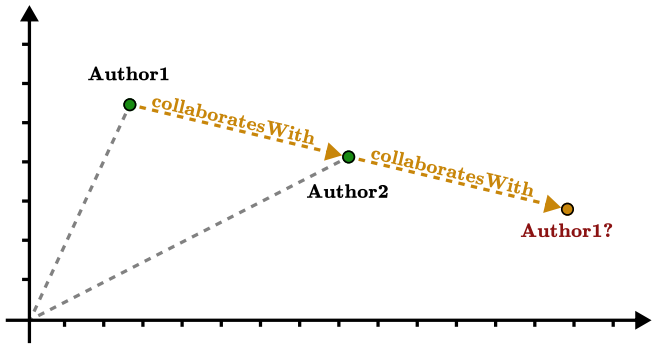

Fig. 2: Representation of a 2D symmetric relationship in a transactional embedding model such as *TransE*.

which imply that $\mathbf{Author1} \approx \mathbf{Author2}$ and that $\mathbf{collaboratesWith} \approx \mathbf{0}$. This struggle is represented in figure 2.

To circumvent this, models such as *RotatE* were developed, which models the relations in a complex vector space, modelling entity embeddings as complex vectors $\mathbf{e} \in \mathbb{C}^k$, and relations also as complex vectors $\mathbf{r} \in \mathbb{C}^k$ in which each component has absolute value equal to 1. It then computes the relationships as

$$\mathbf{head} \odot \mathbf{rel} \approx \mathbf{tail},$$

which, given that a product by a unit complex number is equivalent to a rotation in the complex space, effectively rotates the head vector (hence, the name). With this paradigm, for a relation to be symmetric between two entities, the following must hold:

$$\mathbf{head} \odot \mathbf{rel} = \mathbf{tail} \quad \text{and} \quad \mathbf{tail} \odot \mathbf{rel} = \mathbf{head},$$

which holds only if **rel** corresponds to a rotation of 0º or 180º. In other words, $\mathbf{r} = \pm 1$ (all elements of $\mathbf{r}$ must be either 1 or -1). An example of how *RotatE* would solve a 1-dimensional ($\mathbb{C}^1$) symmetric relation can be seen in figure 3.

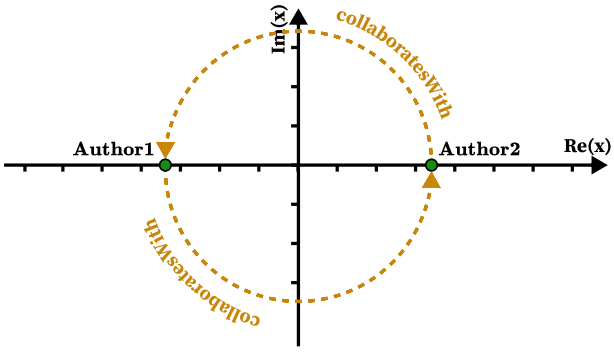

Fig. 3: Representation of a symmetric relationship in $\mathbb{C}^1$ in a rotational embedding model such as *RotatE*. In this figure, $\mathbf{collaboratesWith}$ is the vector $-1 + 0i$.

It's important to note that for *RotatE* to learn that a specific relation is symmetric, the triples in both directions must appear

in the training dataset, and no asymmetric (or antisymmetric) triples must exist that use the same relation.

2) Modeling Unrestricted Symmetric Relations: Among the models presented in the statement, one is especially good for modelling symmetric relations. In fact, it cannot model asymmetric relations at all. We are talking about the *DistMult* model, which instead of "projecting" the embedding of a *tail* in terms of the ones from the *head* and the *relation*, it directly computes the score of each triple (h, r, t) as follows:

$$f(h, r, t) = \langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle = \sum_{i=1}^{k} h_i \cdot r_i \cdot t_i,$$

which is obviously symmetric ($\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle = \langle \mathbf{t}, \mathbf{r}, \mathbf{h} \rangle$). To circumvent the limitation of *DistMult*, model *ComplEx* was developed, which uses complex vectors and the following score function:

$$f(h, r, t) = \Re\left(\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle\right) = \Re\left(\sum_{i=1}^{k} h_i \cdot r_i \cdot \bar{t}_i\right).$$

Where $\bar{t}_i$ is the complex conjugate of the element i of the complex vector $\mathbf{t} \in \mathbb{C}^k$. Note that, in the above function, if we limit $\mathbf{r}$ to the vector sub-space $\mathbb{R}^k \subset \mathbb{C}^k$, we get symmetric relations such as the ones in *DistMult*.

C. Training KGEs

Before exploring applications with KGEs, we have to construct and choose the final model. The base models to consider by the statement are *TransE*, *TransH*, *TransR*, *RotatE*, *DistMult* and *ComplEx*. After studying all of them, and taking into account the computational cost of executing the training, we chose three:

1) **TransE**: TransE will act as our perfect baseline model for two reasons: is one of the simplest models with which we can create the KGEs and we already have it trained from the previous sections. On the other hand, we are not going to use *TransH* (Translation on Hyperplanes) or *TransR* (Translation on Relation-specific Spaces). These two models are generalizations of *TransE* and we want to try other more different models. Additionally, both *TransH* and *TransR* increase training time and parameter complexity.
2) **ComplEx**: Before considering ComplEx, we discarded *DistMult*. While *DistMult* handles only symmetric relations, *ComplEx* introduces asymmetry by taking the complex conjugate of tail vectors. Besides, *ComplEx* is fundamentally different from *TransE*: it operates in a complex-valued space and uses multiplicative interactions rather than translations
3) **RotatE**: RotatE is another model that works in the complex space, like *ComplEx*, but with a different idea: instead of relying on inner products, it represents relations as rotations in the complex plane. This lets it naturally model patterns like symmetry, antisymmetry, and inversion. Compared to *TransE*, iit adds more flexibility, without becoming too complex or heavy to train.

To test the chosen models, we have performed a first 10-epoch training for each three. Next, we compared the results from the link prediciton evalutation to choose the model where we will perform further experiments on the hyperparameters. The evaluation metrics considered are Hits@N, the proportion of of test triples where the correct entity is in the top N results; MRR, averaging the inverse of its rank across all queries and Harmonic Mean Rank, the average rank of the correct answer.

In TABLE III we can see the results where all the metrics point out that **RotatE** is the best performer in the link prediciton.

Model	Hits@10	Hits@3	Hits@1	MRR	Harm. Rank
RotatE	0.0474	0.0369	0.0290	0.0357	28.01
TransE	0.0325	0.0165	0.0102	0.0178	56.25
ComplEx	0.0146	0.0093	0.0056	0.0089	112.02

TABLE III: Performance comparison on citation link prediction.

Once find out that RotatE is the best model in the base configuration, we will play with the hyperparameters to further leverage the benefits of the model and arrive at a configuration that is closer to optimal. The hyperparameters that have been considered for the hypertuning are `embedding_dim`, `learning_rate`, `num_epochs` and `num_negs_per_pos`. In a scientific work with acces to supercomputing or powerful GPUs, it would make sense to tune as much parameters as possible with as much values as possible to find one of the best embedding models. However, the objective of this work is academic. With the limitations of our local environments, computation time limit in Colab and given the fact that every train-evaluation process last almost one hour, we will try two more hyperparameter configurations.

- **`embedding_dim 64`, `num_negs_per_pos 1`**
  A smaller embedding size to make the model faster and lighter. This setting helps us see how well RotatE performs with reduced capacity and whether we can save resources without losing much accuracy. Moreover, we would like to test if embedding the embedding dimension of 64 would lead to under fit or, it would outperform the baseline.
- **`embedding_dim 128`, `num_negs_per_pos 5`**
  Building on the baseline setup, this configuration increases the number of negative samples per positive triple from 1 to 5. Negative sampling plays a critical role in contrastive learning for knowledge graph embedding models. A higher number of negative samples provides the model with more challenging examples, potentially improving generalization and discriminative power.

The result of these experiments are in Table IV. There is not an improvement with the new hyperparameter configurations, but there is also not a big decrease in the performance. For instance, the harmonic rank value of RotatE (64, 1) is very

simililar to our RotatE baseline. Thus, the best model be in between these two or even modifying the other ones. For a future work with more computational resources it would be interesting to keep tuning these and the other parameters combined.

TABLE IV: Comparison of RotatE Variants with Different Embedding Dimensions and Negative Samples

Model (emb, negs)	Hits@10	Hits@3	Hits@1	MRR	Harmonic Rank
RotatE (128, 1)	0.0474	0.0369	0.0290	0.0357	28.01
RotatE (64, 1)	0.0476	0.0339	0.0243	0.0325	30.75
RotatE (128, 5)	0.0386	0.0276	0.0202	0.0268	37.28

After the different experiments the model that will be used for Section III-D is RotatE (128, 1): with the number of training epochs of 10, learning rate of 0.01 and random seed 2025.

D. Exploiting KGEs

Seeing as the model that yielded the best results was *RotatE*, we decided to go with its representations to build something interesting. The important thing to notice is that this model is one of the few (together with *TransE*) whose interaction function is such that given the head embedding and the relation embedding, the tail embedding can be computed. In contrast, models like *DistMul* use interaction functions is such that a prediction must always be *differential*[6].

Specifically, the interaction function of the *RotatE* model is the following:

$$\|\mathbf{h} \odot \mathbf{r} - \mathbf{t}\|,$$

where $\mathbf{h} \odot \mathbf{r}$ can be thought of as a "predicted target embedding". The other thing important to notice is that, given that $\|\mathbf{r}\| = 1$ is a rotation the relation can be "inverted" by simply performing the same rotation in the inverse direction. In other words:

$$\text{Emb}(r^-) = \bar{\mathbf{r}} \Longrightarrow \mathbf{t} \odot \bar{\mathbf{r}} \approx \mathbf{h}$$

With this, we were equipped to predict all kinds of "plausible" embeddings given any set of nodes and relations. For instance, given a paper, compute the most likely embedding of an author ($writesPaper^-$), or a paper that cites it ($paperCites^-$).

1) Reviewer Recommendation System: We decided to develop a system in which, given a **new** paper (one that is not in the knowledge graph),

1) Predicts the most plausible embedding of the new paper, given a set of properties.
2) Predicts the most plausible embedding of a reviewer for such paper.
3) Returns the k authors closest to said embedding, which should be good recommendations of reviewers.

[6]By checking the score between two different embeddings and choosing the one that maximizes it, instead of generating a plausible embedding.

To do so, two scripts were developed:

`create_paper_info.py`

This script prompts the user for some information about the paper required by the algorithm. To do so, at each step, it offers a choice to the user between several available entities. For instance, to choose the publication venue of the paper, it first asks the user if the paper is being published in a Journal, Proceedings or something else — let's say the user responds "Journal" — then, it queries the database to get all available journals, and asks them to identify the one they are publishing at; finally, it prompts the user to choose between all available journal volumes.

After having gathered all required information (list of topics, publication venue, list of authors, list of references, and list of citations, if available), it stores the URNs of all entities in a YAML file.

`recommend_reviewers.py`

This script loads the data as stored by the first one, checks that all entities exist in the database, and performs the following:

1) For each information provided in the file, it obtains its embedding, and tries to map, using the embeddings of known relations, to a potential paper. For instance, if we know that the publication venue is v, we obtain its embedding $\mathbf{v}$, the embedding of relation $isPublishedIn^-$ $\bar{\mathbf{r}}$[7], then compute a plausible paper embedding as $\mathbf{p}' = \mathbf{v} \odot \bar{\mathbf{r}}$.
2) Average all plausible embeddings obtained from all the information, to get the most probable embedding for the new paper.
3) Using a concatenation of $reviewIsAbout^-$ and $writesReview^-$, get the most probable embedding of a plausible reviewer of the new paper.
4) Using heaps and a modified form of L2 distance for complex numbers, compute the top k authors that are most close to the computed embedding. To do so, the script must first obtain a list of all authors in the graph, remove those who wrote the new paper, and filter all entities based on whether they are or not in that set.

2) Goodness of the recommendations: Given that the embeddings obtained are very poor, no significant results were obtained. Authors recommended seldom had anything to do with the topics of the paper.

IV. Conclusions

In this paper we have fully experimented with the possibilities of knowledge graphs by running their different applications on a bibliographic database. First of all, GraphDB proved to

[7]Because we have defined *isPublishedInJournal* and *isPublishedInProceedings* to be subclasses of *isPublishedIn*, we can use the generic class here.

be a powerful platform for storing and querying RDF data, especially when combined with the capabilities of SPARQL.

It was with GraphDB and rdflib that we were able to create and explore the ontology from the definition of the TBOX by their simpler triplets and the integration of SemanticScholar data for the ABOX. Having a total of 625,000 triples has slowed down the execution of some parts of the pipeline. To reflect more specific knowledge constraints, it would have been interesting to have the flexibility offered by other languages such as OWL. Nevertheless, it is impressive the amount of information we have been able to extract by defining the network with single triples.

Then, we have built Knowledge Graph Embeddings (KGEs) that we can later apply to a several machine learning problems. Understanding how the TransE, TransH, TransR, RotatE, DistMult and ComplEx models work mathematically, we have found that, for our project, the three most interesting of them are TransE, RotatE and ComplEx. In this sense, we have trained KGEs with them, and have found that RotatE yielded the best results. Thus, we have performed hyperparameter tuning for RotatE model.

Building on the learned embeddings from the RotatE model, we developed a prototype reviewer recommendation system. This system simulates the process of assigning reviewers to a new paper by first constructing an embedding for the hypothetical paper based on selected properties such as its publication venue, topics, citations, and references, and then using the inverse of relevant relations to compute a plausible embedding for a potential reviewer, and retrieving the authors with the closest embeddings. This method yielded not so good candidates, nevertheless, it serves as a proof of concept for leveraging knowledge graph embeddings for real-world recommendation tasks.

V. Future Work

Although the current reviewer recommendation system demonstrates the feasibility of using knowledge graph embeddings for this task, its results are still far from optimal. A clear direction for improvement is the incorporation of richer semantic and contextual signals. For example, we would like to enhance the model by prioritizing reviewers who have authored multiple papers on similar topics, or those with high citation counts, which would act as an indicator of influence or expertise. Additionally, incorporating more fine-grained information such as institution affiliation, publication year, or co-authorship networks could help the model better assess conflict of interest and reviewer appropriateness.

Furthermore, to improve the quality of the embeddings used in the recommendation system, a more thorough exploration of the hyperparameter space is essential. While our current model tuning was limited to a few configurations due to computational constraints, a systematic search using techniques such as grid search or Bayesian optimization combined with cross-validation could lead to significantly better embeddings. Evaluating models more robustly across validation splits would help prevent overfitting and ensure that the learned representations generalize well across different parts of the graph.

Another avenue would be the use of more advanced methods for generating paper embeddings from partial information, including attention-based aggregation of known entities or pretraining embeddings with additional textual features (e.g., abstracts or titles). Beyond technical refinements, we also aim to define more rigorous evaluation metrics to assess the quality of reviewer recommendations, potentially including human-in-the-loop feedback or comparisons with actual reviewer assignments in public datasets.