## Knowledge Graph Assignment Report

## TBOX Design Methodology

## TBOX modeled key concepts like Paper, Author, Conference, Edition, Review, and Topic as classes. Object properties were defined with domain and range for inference support.

## ABOX Generation Methodology

## RDFLib was used to generate the ABOX based on real and synthetic paper metadata. Entities and relationships were converted to RDF triples and linked via properties from the TBOX.

## Inference Regime

## RDFS reasoning enabled automatic typing of entities based on property usage, such as inferring Reviewer from assignedReviewer.

## Knowledge Graph Summary

## Metric

## Value

## Number of Classes

## 5

## Number of Properties

## 8

## Number of Triples

## See RDF

## Querying the Ontology

## 5.1 Query 1: Reviewers of Each Paper

## This query lists papers and their reviewers using ex:hasReview and ex:assignedReviewer. Type inference is used to classify reviewers.

## PREFIX ex: http://example.org/research/ SELECT ?paper ?review ?reviewer WHERE { ?paper a ex:Paper ; ex:hasReview ?review . ?review ex:assignedReviewer ?reviewer . } LIMIT 20

## 5.2 Query 2: Most Cited Papers with Topics

## This query ranks papers by number of citations and retrieves their associated topics.

## PREFIX ex: http://example.org/research/ SELECT ?paper (COUNT(?citation) AS ?citationCount) (GROUP\_CONCAT(DISTINCT ?topic; separator=", ") AS ?topics) WHERE { ?paper a ex:Paper . OPTIONAL { ?otherPaper ex:cites ?paper . BIND(?otherPaper AS ?citation) } OPTIONAL { ?paper ex:hasKeyword ?topic } } GROUP BY ?paper ORDER BY DESC(?citationCount) LIMIT 10

## Knowledge Graph Embeddings (KGEs)

## 6.1 Data Preparation

## To generate knowledge graph embeddings (KGEs), we first exported our RDF triples from GraphDB in TSV format. We filtered the exported data to include only structurally meaningful predicates relevant for embedding learning. Specifically, we included triples with the following properties:

## ex:hasAuthor

## ex:cites

## ex:hasKeyword

## ex:assignedReviewer

## ex:publishedInJournal

## Properties such as ex:hasAbstract or ex:heldInYear were excluded as they are not useful for structural embedding learning. The filtered triples were saved in TSV format, suitable for PyKEEN.

## Using PyKEEN, we loaded the TSV data and used the built-in functionality to generate stratified training and test sets. This ensures that all types of relationships are well distributed across both datasets, which is crucial for training embedding models effectively.

## 6.2 TransE Model Analysis

## 6.2.1 Predicting Relationships using Embeddings

## We selected a paper entity and used the TransE model to compute which paper it is most likely to cite. This was done by computing the vector h + r\_cites and finding the closest embedding vector. Then, we predicted the author of the cited paper by computing h2 + r\_hasAuthor. We compared the result vectors with all entity embeddings to find the closest matches in terms of Euclidean distance. This demonstrated how TransE can be interpreted geometrically.

## The prediction results were saved in a text file named 'kge\_results.txt'. The TransE model and all computations were implemented in the script 'PyKEEN.py'.

## 6.2.2 Limitations of TransE and Alternatives

## Multi-relational Embedding Conflict in TransE

## Given the triples: (Author1, writes, Paper1), (Author2, writes, Paper1), and (Author1, writes, Paper2), TransE must position the 'writes' relation as a vector that simultaneously connects multiple authors to the same paper and one author to multiple papers. This creates a conflict, as TransE assumes a single vector translation for a relation. Therefore, it cannot perfectly model these One-to-Many or Many-to-One relationships without sacrificing embedding accuracy.

## Model Designed to Address These Issues

## The TransH model was created to partially address these problems. Instead of enforcing a single translation vector in one space, TransH projects entities onto a relation-specific hyperplane and then applies the translation. This allows for different embeddings per relation context and helps model complex relationships more flexibly.

## Symmetry in TransE

## If the relation 'collaboratesWith' is symmetric, then (Author1, collaboratesWith, Author2) and (Author2, collaboratesWith, Author1) should ideally have the same score. However, in TransE, the scoring function is based on vector distance h + r ≈ t. This is directional and does not guarantee symmetry unless a trivial or degenerate solution is learned where the embedding vectors or translation vectors collapse.

## Why TransE Fails to Model Symmetry

## In a 2D space, if Author1 is at position A1 and Author2 at A2, then for both triples to be valid:

## A1 + r ≈ A2 and A2 + r ≈ A1

## Solving for r leads to: A2 - A1 ≈ r and A1 - A2 ≈ r → contradiction unless A1 ≈ A2 or r ≈ 0

## This creates a degenerate solution and shows why symmetry is not inherently captured by TransE.

## Symmetry with RotatE

## RotatE models relations as rotations in complex space. A relation is symmetric if rotating by r from Author1 gives Author2, and rotating by r from Author2 gives Author1. This is achieved when r^2 = 1 (i.e., rotation angle of 180° or π), leading to reversible paths. In 2D complex space, symmetry is modeled by complex conjugation or reflection, which RotatE can naturally represent.

## Model Allowing Symmetry Without Constraints

## The ComplEx model allows symmetry and antisymmetry by operating in complex vector space with Hermitian inner products. Its scoring function is not dependent on directional vector differences and supports both symmetric and asymmetric relations.

## 6.3 Training and Comparing KGE Models

## To compare different Knowledge Graph Embedding (KGE) models, we trained and evaluated four commonly used models using PyKEEN:

## TransE (embedding\_dim=50)

## ComplEx (embedding\_dim=100)

## DistMult (embedding\_dim=100)

## RotatE (embedding\_dim=100)

## All models were trained for 5 epochs using the sLCWA training loop and the basic negative sampling strategy. The training was performed on the same set of triples exported from our KG in TSV format.

## We evaluated the models using standard metric Mean Reciprocal Rank (MRR) to assess link prediction quality. Here are the results:

## TransE: MRR = 0.0338

## ComplEx: MRR = 0.0014

## DistMult: MRR = 0.0052

## RotatE: MRR = 0.3137

## RotatE significantly outperformed the others in MRR and was thus selected for further exploitation.

## 6.4 Exploiting KGEs

## For the final task, we applied the learned RotatE embeddings to explore knowledge graph structure. Specifically, we selected a paper entity and computed similar entities by comparing embedding distances. Additionally, we evaluated the model to assess its link prediction quality.

## Selected Paper: http://example.org/research/paper\_000f4079fb3f5463ce7f5e566b541fec68a91f02

## Closest Predicted Entity: http://example.org/research/paper\_000f4079fb3f5463ce7f5e566b541fec68a91f02

## Evaluation Summary (MRR and Hits@k):

## Mean Reciprocal Rank (MRR): 0.0865

## Hits@1: 3.05%

## Hits@3: 9.67%

## Hits@5: 13.76%

## Hits@10: 20.08%

## These values indicate moderate predictive power with especially good recall at top-10 predictions.

## The script 'PyKEEN\_Exploitation.py' was used to perform these tasks and generate the output.

## 6.2 TransE Model Analysis

### 6.2.1 Predicting Relationships using Embeddings

We selected a paper entity and used the TransE model to compute which paper it is most likely to cite. This was done by computing the vector h + r\_cites and finding the closest embedding vector. Then, we predicted the author of the cited paper by computing h2 + r\_hasAuthor. We compared the result vectors with all entity embeddings to find the closest matches in terms of Euclidean distance. This demonstrated how TransE can be interpreted geometrically.

The prediction results were saved in a text file named 'kge\_results.txt'. The TransE model and all computations were implemented in the script 'PyKEEN.py'.

### 6.2.2 Limitations of TransE and Alternatives

1. \*\*Multi-relational Embedding Conflict in TransE\*\*

Given the triples: (Author1, writes, Paper1), (Author2, writes, Paper1), and (Author1, writes, Paper2), TransE must position the 'writes' relation as a vector that simultaneously connects multiple authors to the same paper and one author to multiple papers. This creates a conflict, as TransE assumes a single vector translation for a relation. Therefore, it cannot perfectly model these One-to-Many or Many-to-One relationships without sacrificing embedding accuracy.

2. \*\*Model Designed to Address These Issues\*\*

The TransH model was created to partially address these problems. Instead of enforcing a single translation vector in one space, TransH projects entities onto a relation-specific hyperplane and then applies the translation. This allows for different embeddings per relation context and helps model complex relationships more flexibly.

3. \*\*Symmetry in TransE\*\*

If the relation 'collaboratesWith' is symmetric, then (Author1, collaboratesWith, Author2) and (Author2, collaboratesWith, Author1) should ideally have the same score. However, in TransE, the scoring function is based on vector distance h + r ≈ t. This is directional and does not guarantee symmetry unless a trivial or degenerate solution is learned where the embedding vectors or translation vectors collapse.

4. \*\*Why TransE Fails to Model Symmetry\*\*

In a 2D space, if Author1 is at position A1 and Author2 at A2, then for both triples to be valid:

A1 + r ≈ A2 and A2 + r ≈ A1

Solving for r leads to: A2 - A1 ≈ r and A1 - A2 ≈ r → contradiction unless A1 ≈ A2 or r ≈ 0

This creates a degenerate solution and shows why symmetry is not inherently captured by TransE.

5. \*\*Symmetry with RotatE\*\*

RotatE models relations as rotations in complex space. A relation is symmetric if rotating by r from Author1 gives Author2, and rotating by r from Author2 gives Author1. This is achieved when r^2 = 1 (i.e., rotation angle of 180° or π), leading to reversible paths. In 2D complex space, symmetry is modeled by complex conjugation or reflection, which RotatE can naturally represent.

6. \*\*Model Allowing Symmetry Without Constraints\*\*

The ComplEx model allows symmetry and antisymmetry by operating in complex vector space with Hermitian inner products. Its scoring function is not dependent on directional vector differences and supports both symmetric and asymmetric relations.

## 6.3 Training and Comparing KGE Models

To compare different Knowledge Graph Embedding (KGE) models, we trained and evaluated four commonly used models using PyKEEN:

* - TransE (embedding\_dim=50)  
  - TransH (embedding\_dim=50)  
  - ComplEx (embedding\_dim=100)  
  - DistMult (embedding\_dim=100)

All models were trained for 100 epochs using the sLCWA training loop and the basic negative sampling strategy. The training was performed on the same set of triples exported from our KG in TSV format.

We evaluated the models using standard metrics: Mean Rank (MR), Mean Reciprocal Rank (MRR), Hits@1, and Hits@10. These metrics help measure the ranking quality of predicted entities for each test triple.

The experiment results were saved in 'kge\_experiment\_results.csv'. Based on these results, we selected ComplEx as the final model for downstream use. It achieved the highest MRR and Hits@10, indicating strong performance in link prediction tasks.

## 6.4 Exploiting KGEs

For the final task, we applied the learned ComplEx embeddings to an unsupervised machine learning task: clustering authors based on their embedding vectors. This helps to identify groups of researchers that share similar structural roles in the knowledge graph.

We filtered the set of entity embeddings to include only authors, then applied the K-Means clustering algorithm with k=3. To visualize the clusters, we reduced the embedding dimensionality using PCA and plotted the results in a 2D space.

The Python script for this task is included as 'PyKEEN\_Exploitation.py'. It generates and saves the visualization 'kge\_author\_clusters.png', highlighting the emergent structure among authors in the embedding space.

This task demonstrates how KGEs can be used for tasks like author profiling, collaboration prediction, or scientific community detection.