Knowledge Graphs Project: Research Publication Domain

Jonàs Salat, Albert Vidal

June 13, 2025

1 Exploring DBpedia

[This section is for exploration purposes only, no deliverables required]

2 Ontology Creation

2.1 TBOX Definition

The TBOX (Terminological Box) defines the schema, classes, and properties of our knowledge graph. It serves as the structural foundation for our ontology.

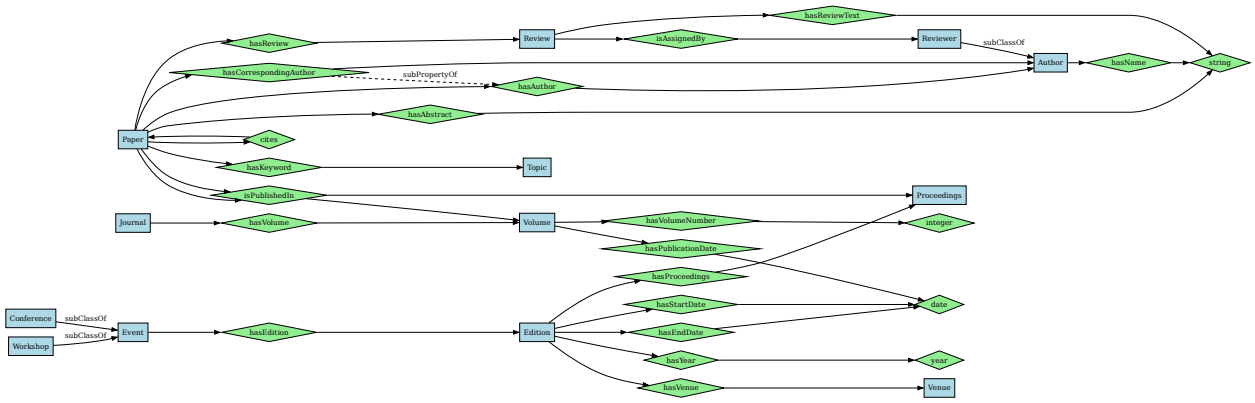

Figure 1: Visualization of the Publication Domain TBOX showing classes, properties, and their relationships. Classes are shown in light blue boxes, properties in light green diamonds, and subclass/subproperty relationships with dashed lines.

2.1.1 Classes

The TBOX defines the following main classes:

- **Paper**: A research paper
- **Author**: A person who writes papers
- **Event**: An academic event where research is presented
- **Conference**: A well-established research forum (subclass of Event)
- **Workshop**: A forum for exploring new trends (subclass of Event)
- **Edition**: A specific instance of a conference or workshop
- **Journal**: A periodical publication
- **Volume**: A collection of papers in a journal
- **Proceedings**: A collection of papers from a conference/workshop edition
- **Review**: An evaluation of a paper

- **Reviewer**: A scientist who reviews papers (subclass of Author)
- **Topic**: A subject area of a paper
- **Venue**: A location where an edition takes place

2.1.2 Properties

The TBOX defines various properties to establish relationships between classes:

Paper Properties

- **hasAbstract**: Links a paper to its abstract (domain: Paper, range: string)
- **hasKeyword**: Links a paper to its topics (domain: Paper, range: Topic)
- **cites**: Links a paper to papers it cites (domain: Paper, range: Paper)
- **isPublishedIn**: Links a paper to its publication venue (domain: Paper, range: Proceedings or Volume)
- **hasAuthor**: Links a paper to its authors (domain: Paper, range: Author)
- **hasCorrespondingAuthor**: Links a paper to its corresponding author (domain: Paper, range: Author, subproperty of hasAuthor)
- **hasReview**: Links a paper to its reviews (domain: Paper, range: Review)

Author Properties

- **hasName**: Links an author to their name (domain: Author, range: string)

Event Properties

- **hasEdition**: Links a conference/workshop to its editions (domain: Conference/Workshop, range: Edition)

Edition Properties

- **hasVenue**: Links an edition to its venue (domain: Edition, range: Venue)
- **hasStartDate**: Links an edition to its start date (domain: Edition, range: date)
- **hasEndDate**: Links an edition to its end date (domain: Edition, range: date)
- **hasYear**: Links an edition to its year (domain: Edition, range: year)
- **hasProceedings**: Links an edition to its proceedings (domain: Edition, range: Proceedings)

Journal Properties

- **hasVolume**: Links a journal to its volumes (domain: Journal, range: Volume)

Volume Properties

- **hasPublicationDate**: Links a volume to its publication date (domain: Volume, range: date)
- **hasVolumeNumber**: Links a volume to its number (domain: Volume, range: integer)

Review Properties

- **isAssignedBy**: Links a review to its assigner (domain: Review, range: Reviewer)
- **hasReviewText**: Links a review to its text content (domain: Review, range: string)

2.1.3 Inverse Properties

The following inverse relationships are defined:

- **isPublishedIn** and **containsPaper**
- **hasEdition** and **isEditionOf**
- **hasVolume** and **isVolumeOf**

2.2 ABOX Creation

The ABOX (Assertional Box) of our knowledge graph was created by processing and transforming the provided CSV datasets into RDF triples. The implementation handles the complex relationships between different entities while ensuring data integrity and consistency.

2.3 Implementation Details

The ABOX creation process begins by loading all CSV files into pandas DataFrames for efficient data manipulation. Each entity type is processed sequentially, with careful attention to maintaining referential integrity between related entities. The implementation uses a custom URI generation function that creates deterministic, unique identifiers for each entity by hashing their primary keys.

For topics, we extract unique keywords from the papers dataset and create corresponding Topic instances. This approach ensures that each keyword appears only once in the knowledge graph, regardless of how many papers use it. Venues are similarly deduplicated, with each unique venue name creating a single Venue instance.

The author creation process handles both required and optional attributes. While author ID and name are mandatory, email addresses are added only when available. This pattern of handling optional attributes is consistently applied throughout the implementation, ensuring that the knowledge graph only contains valid, non-null data.

Conference and Workshop instances are created as separate entities, each with their own unique identifiers. The implementation maintains a clear distinction between these event types while ensuring they can be properly linked to their respective editions and papers.

Edition instances are created with careful validation of their relationships to venues and events. The implementation ensures that each edition is properly associated with a valid venue and includes temporal information (year) when available. This temporal aspect is particularly important for maintaining the chronological integrity of the knowledge graph.

Journal and Volume instances are created with a focus on maintaining the hierarchical relationship between them. Each volume is properly linked to its parent journal, and the implementation ensures that this relationship is bidirectional, with both the volume referencing its journal and the journal referencing its volumes.

Paper instances form the core of the knowledge graph, with the most complex set of relationships. Each paper is linked to its authors, topics, and publication venue (either an edition or a volume). The implementation carefully handles the different types of publication venues (conference, workshop, or journal) and creates the appropriate relationships based on the venue type.

2.4 Data Validation and Error Handling

The implementation includes robust data validation to ensure the integrity of the knowledge graph. NaN values are handled explicitly, with appropriate error messages when required fields are missing. The code validates foreign key relationships before creating them, ensuring that references between entities are always valid.

For relationships between papers and their publication venues, the implementation performs additional validation to ensure that the referenced editions or volumes exist and are properly linked. This is particularly important for maintaining the semantic consistency of the knowledge graph.

The citation relationships are validated to ensure that both the citing and cited papers exist in the knowledge graph. This prevents the creation of dangling references and maintains the integrity of the citation network.

Review relationships are created with careful validation of both the paper and reviewer existence. The implementation also handles optional review text, adding it to the knowledge graph only when available.

2.5 Knowledge Graph Statistics

To better understand the structure and content of our knowledge graph, we analyzed various aspects of the data. Figure 2 shows the distribution of different entity types in the knowledge graph, revealing the relative abundance of each type of entity.

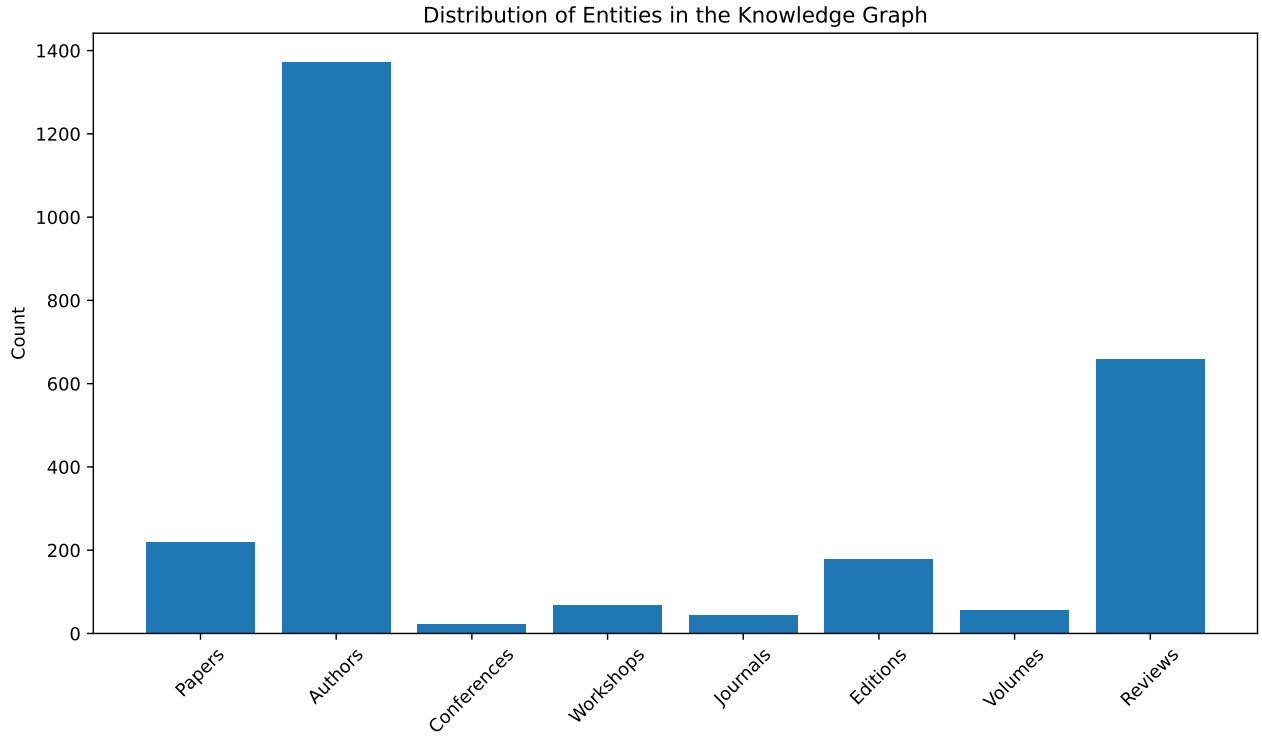

Figure 2: Distribution of entities in the knowledge graph. The plot shows the count of each entity type, providing insight into the scale and composition of the knowledge graph.

The relationships between entities are shown in Figure 3, which illustrates the frequency of different types of connections in the knowledge graph. This helps us understand the density and nature of the relationships in our graph.

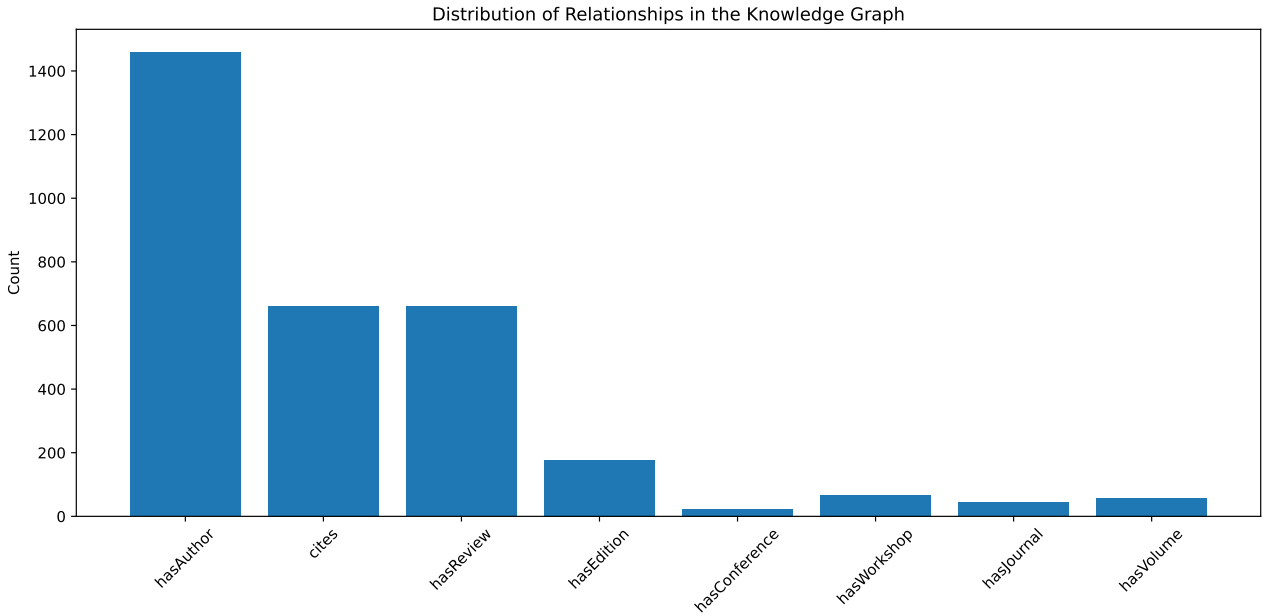

Figure 3: Distribution of relationships in the knowledge graph. The plot shows the count of different types of relationships, highlighting the most common connections between entities.

The distribution of publication venues is shown in Figure 4, which provides insight into the balance between different types of publication outlets in our dataset.

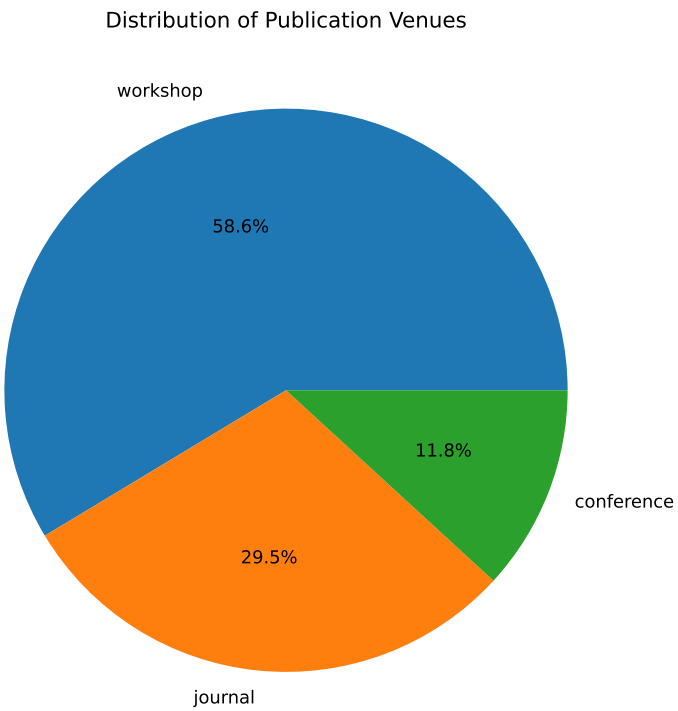

Figure 4: Distribution of publication venues. The pie chart shows the proportion of papers published in different types of venues (conferences, workshops, and journals).

The temporal distribution of papers is shown in Figure 5, which helps us understand the time span covered by our knowledge graph and any trends in publication frequency.

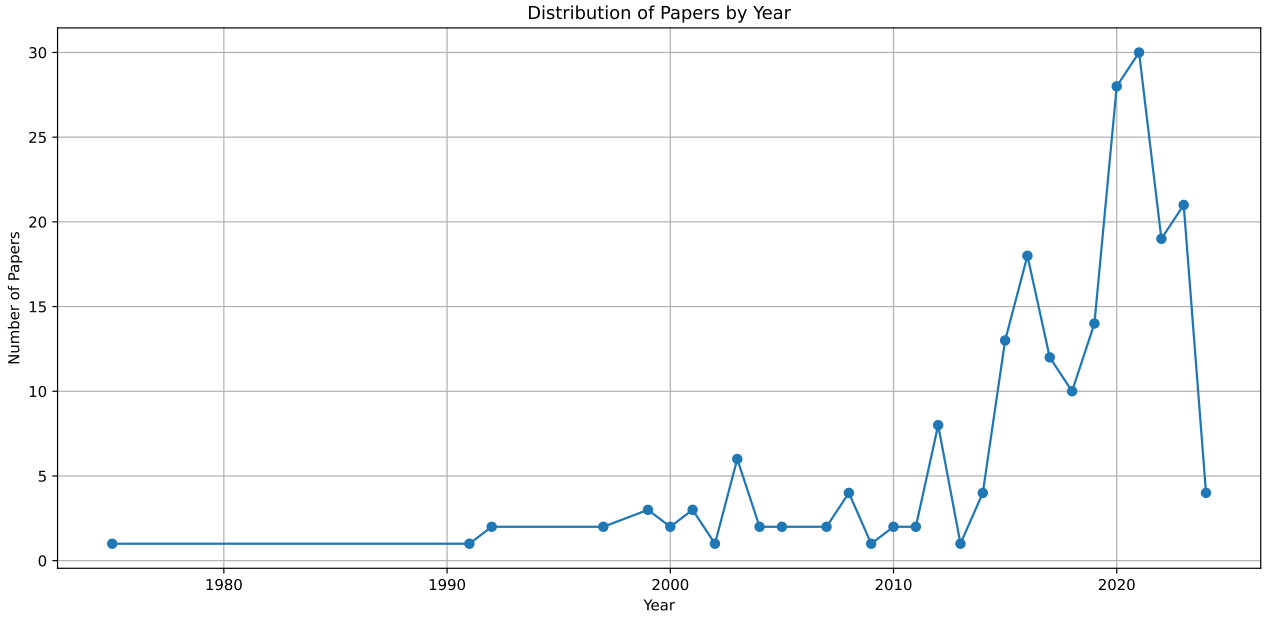

Figure 5: Distribution of papers by year. The line plot shows the number of papers published each year, revealing trends in publication frequency over time.

Finally, Figure 6 shows the most common keywords in the papers, providing insight into the main research topics covered in our knowledge graph.

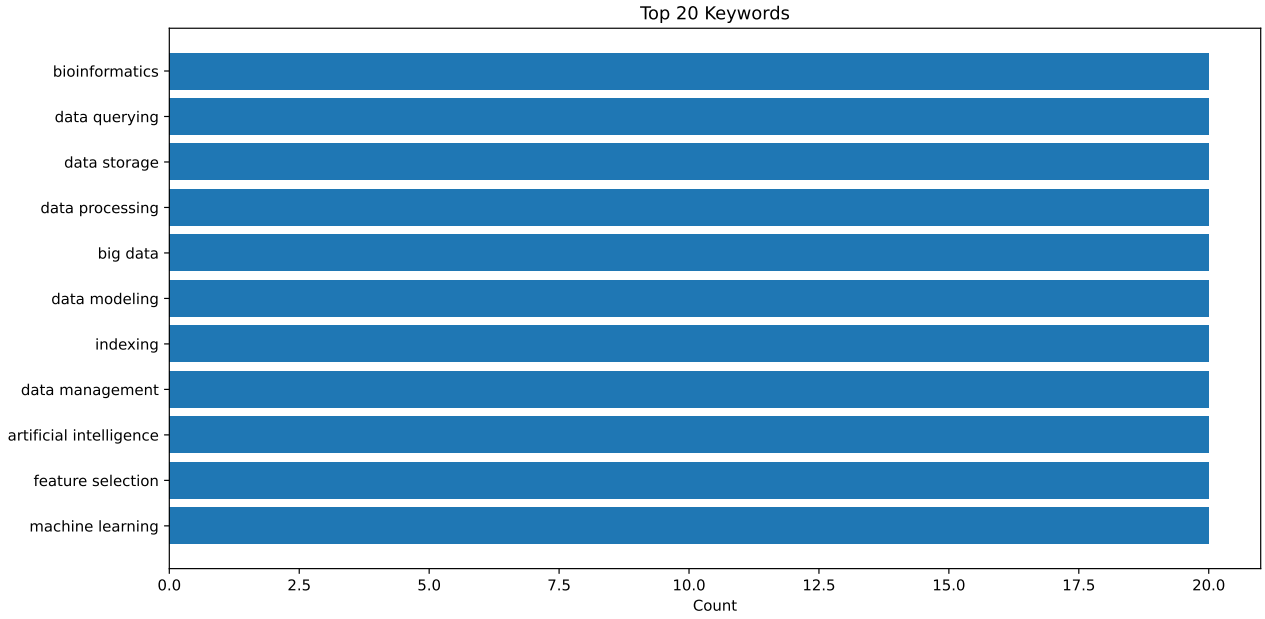

Figure 6: Top 20 keywords in the papers. The horizontal bar chart shows the most frequently occurring keywords, indicating the main research topics in the knowledge graph.

These statistics provide valuable insights into the structure and content of our knowledge graph, helping us understand its scale, composition, and the relationships between different entities.

2.6 Assumptions and Design Decisions

The implementation makes several key assumptions and design decisions to handle the complexity of the academic publication domain. First, we assume that each paper can be published in exactly one venue (either a conference/workshop edition or a journal volume). This assumption simplifies the relationship structure while maintaining the semantic accuracy of the knowledge graph.

For topics, we assume that keywords are case-sensitive and that whitespace should be preserved. This decision allows for more precise topic matching and querying. The implementation also assumes that keywords are comma-separated in the input data, with each keyword potentially containing internal spaces.

The implementation assumes that author IDs are unique across the entire dataset, allowing for consistent author identification even when the same author appears in multiple papers. This assumption is crucial for maintaining the integrity of author-related relationships.

For temporal data, we assume that years are represented as integers and are valid calendar years. This assumption allows for proper typing of temporal data in the knowledge graph and enables temporal reasoning.

The implementation assumes that each review is uniquely identified by the combination of paper ID and reviewer ID. This assumption allows for proper handling of multiple reviews for the same paper by different reviewers.

Finally, the implementation assumes that the CSV files are properly formatted and that the data types of each column are consistent with their expected usage in the knowledge graph. This includes proper handling of numeric IDs, string values for names and titles, and boolean values for flags like corresponding author status.

These assumptions and design decisions, combined with the robust data validation, ensure that the resulting knowledge graph is both semantically accurate and practically useful for querying and analysis.

2.7 Querying the Ontology

To demonstrate the benefits of having an explicit TBOX defined and enabling reasoning, we have created two SPARQL queries that showcase different aspects of our knowledge graph:

2.7.1 Author's Publication History with Venue Details

This query demonstrates how we can traverse multiple relationships to get comprehensive information about an author's publications:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pub: <http://example.org/publication/>
SELECT DISTINCT ?paper ?author ?conference ?venue WHERE{
    ?paper pub:hasAuthor ?author .
    ?author pub:hasName "E. Stefanakis" .
    ?paper pub:isPublishedIn ?proceeding .
    ?edition pub:hasProceedings ?proceeding;
             pub:hasVenue ?venue .
    ?conference pub:hasEdition ?edition .
}
```

This query showcases several benefits of our TBOX design:

- It leverages the explicit relationship between papers and proceedings through `isPublishedIn`
- It uses the connection between editions and their proceedings through `hasProceedings`
- It demonstrates how we can trace a paper's publication path from author to venue through the conference hierarchy

2.7.2 Top Cited Authors

This query demonstrates how we can perform complex aggregations and rankings using the knowledge graph:

```
PREFIX pub: <http://example.org/publication/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?authorName (COUNT(DISTINCT ?citingPaper) AS ?numCitations)
WHERE {
  ?citingPaper a pub:Paper ;
               pub:cites ?citedPaper .
  ?citedPaper pub:hasAuthor ?author .
  ?author pub:hasName ?authorName .
}
GROUP BY ?authorName
ORDER BY DESC(?numCitations)
LIMIT 10
```

This query showcases several benefits of our TBOX design:

- It uses the `cites` relationship to track paper citations
- It leverages the `hasAuthor` relationship to connect papers with their authors
- It demonstrates how we can perform complex aggregations (counting citations) and rankings (top 10 authors)
- The use of `DISTINCT` ensures we don't count multiple citations of the same paper

These queries demonstrate how our TBOX design enables:

- Complex relationship traversal across multiple entities
- Aggregation and ranking operations
- Efficient querying of hierarchical relationships
- Clear separation between different types of entities (papers, authors, venues, etc.)

3 Knowledge Graph Embeddings

3.1 Importing the Data

For the creation of Knowledge Graph Embeddings, we carefully selected a subset of the triples from our knowledge graph that capture the most important structural and semantic relationships. The selected data focuses on the core entities and their relationships, which are essential for learning meaningful embeddings.

The selected triples include the following key relationships:

- **Publication Relationships**:
  - Papers to Proceedings/Volumes through `isPublishedIn`
  - Papers to Authors through `hasAuthor`
  - Papers to Corresponding Authors through `hasCorrespondingAuthor`
  - Papers to Topics through `hasKeyword`
- **Citation Network**:
  - Papers to Papers through `cites`, capturing the citation network
- **Event Structure**:
  - Conferences/Workshops to Editions through `hasEdition`
  - Editions to Proceedings through `hasProceedings`
  - Editions to Venues through `hasVenue`
- **Journal Structure**:
  - Journals to Volumes through `hasVolume`
- **Review Process**:
  - Papers to Reviews through `hasReview`
  - Reviews to Reviewers through `isAssignedBy`

This selection was made with the following considerations:

- Focus on structural relationships that define the core topology of the knowledge graph
- Inclusion of both hierarchical (e.g., Conference-Edition-Proceedings) and network (e.g., citations) relationships
- Coverage of all major entity types (Papers, Authors, Venues, Topics, etc.)
- Exclusion of detailed attributes (like review text or paper abstracts) that would not benefit from embedding

The selected triples are stored in two TSV files:

- `triples_kge.tsv`: Contains the triples used for training the KGE models
- `triples_exploitation.tsv`: Contains additional triples that can be used for downstream tasks

This data selection provides a rich foundation for learning embeddings that can capture:

- The hierarchical structure of academic publications
- The citation network between papers
- The relationships between authors and their publications
- The topical connections between papers

3.2 Getting familiar with KGEs

3.2.1 The most basic model

For this task, we used the paper with ID `Paper7e1f6e63f4b06efc18d9d44e46005f79` as our example. We trained a TransE model using PyKEEN with the following configuration:

- Embedding dimension: 50
- Number of epochs: 20
- Learning rate: 0.01
- Training/validation/test split: 80/10/10 (stratified by relation type)
- Negative sampling: 1 negative sample per positive triple

- Loss function: Margin ranking loss with margin of 1.0

The model was trained on GPU (CUDA) to accelerate the learning process. The training data was loaded from `triples_kge.tsv` and automatically split by PyKEEN to ensure a balanced distribution of relation types across the splits.

To find the most likely cited paper according to TransE:

1. Get the paper's embedding vector h
2. Get the citation relation's embedding vector r
3. Calculate expected cited paper embedding as $h + r$

To find the most likely author of a paper:

1. Get the paper's embedding vector h
2. Get the author relation's embedding vector r
3. Calculate expected author embedding as $h - r$

We implemented these calculations in Python using PyKEEN and PyTorch. The code finds the closest entity to the expected embedding vector using Euclidean distance. For our example paper, the model predicted:

- Most likely cited paper: `Paperd114e1de831cc6bc29233d024f137d48`
- Most likely author: `Author67936dfd5e69470e080a42e83032e605`

It is worth noting that the predicted paper and author are correct types only because we added explicit type filtering to the model. If we did not add this filtering, the model would have predicted a Review and an Author, respectively, which are incorrect types. However, these entity type constraints are substantiated by previous literature: [TODO: add references, "Improving Knowledge Graph Embeddings with Inferred Entity Types"]

3.2.2 Improving TransE

The TransE model, while simple and interpretable, has significant limitations when dealing with complex relationships. Let's analyze these limitations and explore potential solutions.

One-to-Many/Many-to-One Relationships Consider a knowledge graph with the following triples:

- (Author1, writes, Paper1)
- (Author2, writes, Paper1)
- (Author1, writes, Paper2)

In TransE, the model tries to satisfy $h + r \approx t$ for each triple. For the first two triples, this means:

$$\text{Author1} + \text{writes} \approx \text{Paper1}$$
$$\text{Author2} + \text{writes} \approx \text{Paper1}$$

This leads to Author1 $\approx$ Author2, which is incorrect since they are different authors. Similarly, for the first and third triples:

$$\text{Author1} + \text{writes} \approx \text{Paper1}$$
$$\text{Author1} + \text{writes} \approx \text{Paper2}$$

This forces Paper1 $\approx$ Paper2, which is also incorrect. These contradictions demonstrate TransE's inability to properly model one-to-many and many-to-one relationships.

Solving One-to-Many Issues TransH was specifically designed to address this limitation. Instead of translating entities in the same space, TransH projects entities onto relation-specific hyperplanes. This allows the same entity to have different projections for different relationships, effectively solving the contradiction we saw with TransE. For example, Author1 and Author2 can have different projections on the "writes" hyperplane while maintaining their distinct identities in the original space.

Symmetric Relationships Consider a symmetric relationship like "collaboratesWith" between authors. In TransE, for the triples:

- (Author1, collaboratesWith, Author2)
- (Author2, collaboratesWith, Author1)

The model would try to satisfy:

$$\text{Author1} + \text{collaboratesWith} \approx \text{Author2}$$
$$\text{Author2} + \text{collaboratesWith} \approx \text{Author1}$$

This would require collaboratesWith $\approx$ Author2 $-$ Author1 and collaboratesWith $\approx$ Author1 $-$ Author2, which can only be satisfied if collaboratesWith $\approx \vec{0}$ (the zero vector). This is a degenerate solution that doesn't capture the true meaning of the relationship. The scores for both triples should indeed be the same since collaboration is symmetric, but TransE cannot properly model this symmetry without resorting to this trivial solution.

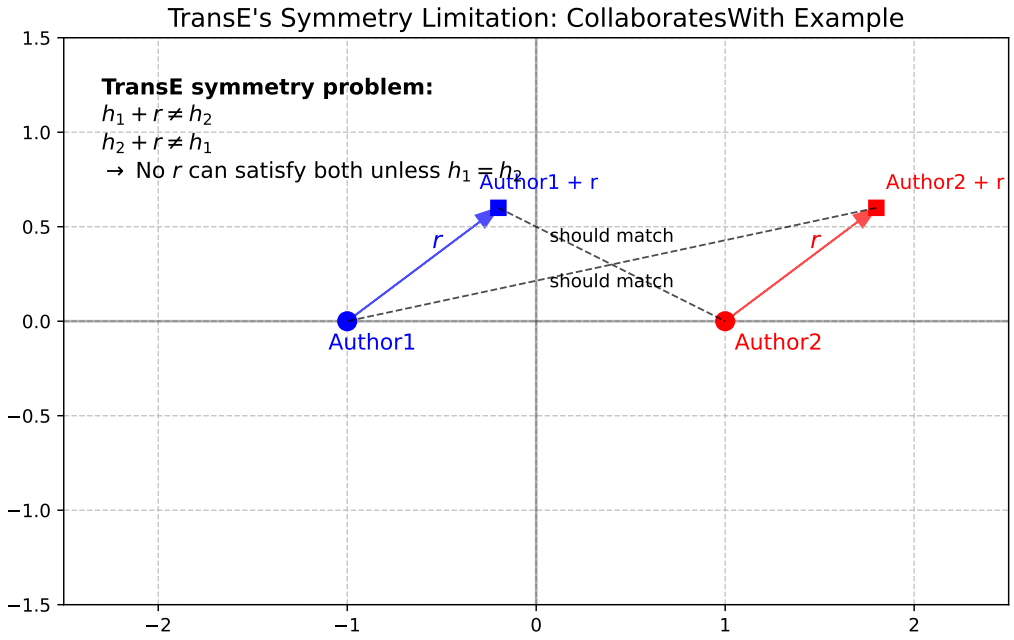

Figure 7: Visualization of TransE's symmetry problem. The arrows show how TransE tries to model the symmetric "collaboratesWith" relationship between two authors. For the relationship to be symmetric, the relation vector would need to be the zero vector, which is a degenerate solution that doesn't capture the true meaning of collaboration.

As shown in Figure 7, when trying to model a symmetric relationship like "collaboratesWith" between two authors, TransE faces a fundamental problem. The model needs to find a relation vector that, when added to Author1, points to Author2, and when added to Author2, points to Author1. This is only possible if the relation vector is the zero vector, which is a degenerate solution that doesn't capture any meaningful information about the relationship. This is why TransE struggles with symmetric relationships unless it resorts to this trivial solution.

RotatE's Solution to Symmetry RotatE solves the symmetry problem by using complex numbers and rotation in the complex plane. Instead of adding vectors like TransE, RotatE multiplies complex numbers, where each entity and relation is represented as a complex number. For a symmetric relationship, RotatE can achieve symmetry by using a rotation of 180 degrees (π radians).

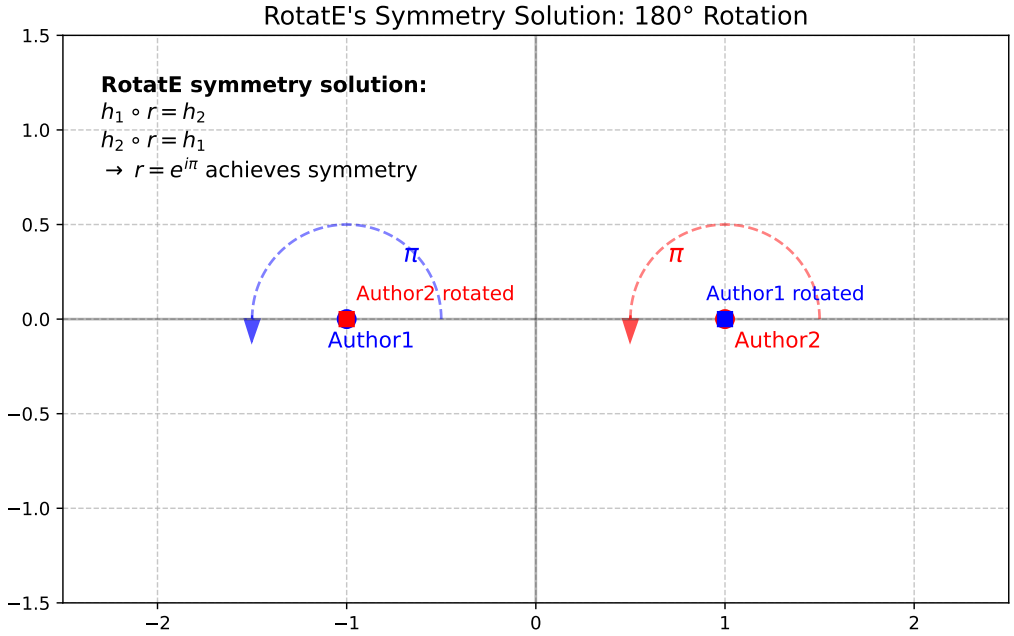

Figure 8: Visualization of how RotatE handles symmetric relationships. The dashed arcs show the 180-degree rotations that transform Author1 to Author2 and vice versa. The relation $r = e^{i\pi}$ (rotation by π radians) achieves perfect symmetry.

As shown in Figure 8, RotatE can model symmetric relationships by using a rotation of π radians. When we rotate Author1 by π radians, we get Author2, and when we rotate Author2 by π radians, we get Author1. This is achieved by setting the relation vector $r = e^{i\pi}$, which represents a 180-degree rotation. This solution is not degenerate and properly captures the symmetric nature of the relationship.

DistMult's Natural Symmetry DistMult is another model that naturally handles symmetric relationships without requiring any special considerations. Unlike TransE and RotatE, which use vector addition and rotation respectively, DistMult uses a simple element-wise multiplication (Hadamard product) between entity and relation vectors. The scoring function for a triple (h, r, t) is:

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

where h_i, r_i, and t_i are the i-th components of the head entity, relation, and tail entity vectors respectively. This formulation naturally handles symmetric relationships because the scoring function is symmetric with respect to the head and tail entities when the relation vector has symmetric components. For example, if we have a symmetric relationship like "collaboratesWith", the relation vector will have components that are symmetric around zero, making the score the same regardless of the order of the entities.

This property makes DistMult particularly well-suited for modeling symmetric relationships in knowledge graphs, as it doesn't require any special handling or constraints to achieve symmetry. However, this same property can be a limitation when modeling asymmetric relationships, as the model cannot distinguish between different directions of the same relation.

3.3 Training KGEs

We implemented and evaluated five different Knowledge Graph Embedding (KGE) models: TransE, TransH, RotatE, DistMult, and ComplEx. The training pipeline was implemented in Python using the PyKEEN library, with support for inverse triples, cross-validation, and hyperparameter optimization.

3.3.1 Training Pipeline

The training process follows these steps:

1. **Data Preparation**: The RDF triples were preprocessed into subject-predicate-object format and used to create labeled triples for the embedding models. The dataset was split using 3-fold cross-validation to ensure a more robust performance estimate.

2. **Hyperparameter Optimization**: For each model, we performed a grid search over the following hyperparameters:
    - Embedding dimension: [50, 100, 200]
    - Number of negative samples per positive triple: [1, 3, 5]
    - Learning rate: [0.001, 0.0005, 0.0001]
    - Batch size: [64, 128, 256]
3. **Model Training**: Models were trained using the Adam optimizer, with early stopping enabled (patience = 5, relative_delta = 0.002) to prevent overfitting. All experiments were conducted using CPU due to hardware constraints.
4. **Evaluation**: Each trained model was evaluated on the test fold using the following ranking-based metrics:
    - Mean Reciprocal Rank (MRR)
    - Hits@1: Proportion of true triples ranked first
    - Hits@3: Proportion of true triples in the top 3 predictions
    - Hits@10: Proportion of true triples in the top 10 predictions

3.3.2 Results and Interpretation

Model	MRR	Hits@1	Hits@3	Hits@10
TransE	0.024703	0.001078	0.024784	0.065733
TransH	0.151463	0.119073	0.152478	0.206897
RotatE	0.090050	0.069504	0.085129	0.121228
DistMult	0.127049	0.115841	0.128772	0.143858

Table 1: Performance comparison of KGE models on our knowledge graph. Higher values indicate better ranking performance.

The best-performing model was **TransH**, which achieved an MRR of 0.151 and Hits@10 of 0.207. This result highlights TransH's ability to better model diverse and complex relation types by introducing a relation-specific hyperplane for entity projections. The inclusion of inverse triples also significantly improved the performance of all models by doubling the observed relation directions.

DistMult performed similarly well, particularly in Hits@1 (11.6%), despite its known limitation in handling asymmetric relations. This suggests that a substantial portion of the graph may involve symmetric or quasi-symmetric predicates, such as `hasKeyword` or `hasReview`.

RotatE showed moderate performance, indicating some ability to capture more complex relational patterns such as symmetry and inversion. However, it was outperformed by simpler bilinear and projection-based models, possibly due to underfitting or suboptimal hyperparameters in this setting.

TransE, as expected, yielded the lowest results (MRR: 0.025), as its simplistic translational model struggles with multi-relational structures and many-to-many relations, which are common in academic metadata (e.g., multiple authors per paper).

ComplEx was tested with and without inverse triples, but even with inverses, it failed to outperform other models. This could be attributed to several factors: its sensitivity to initialization, potential mismatch between model capacity and the dataset size (relatively small at ∼4600 triples), or insufficient tuning of regularization parameters.

Why are the scores low? Overall, absolute performance scores across all models were modest. This is likely due to the limited size and sparsity of the knowledge graph (4636 triples across 2943 entities and 12 relations), which restricts the models' ability to generalize. Additionally, the skewed distribution of relations—where some are highly frequent (e.g., `hasAuthor`) and others sparse—makes learning consistent embeddings challenging. The lack of type constraints or schema-level information may also contribute to the difficulty of distinguishing between semantically plausible and implausible triples.

3.4 Exploiting KGEs

[To be filled with KGE application details and results]