From Top-1 to Top-10: A Two-Stage Entity Alignment Framework for Knowledge Graphs

Table of Contents

Introduction

Entity alignment is a critical task in knowledge graph research, focusing on identifying and linking equivalent entities across different knowledge graphs. Accurate entity alignment facilitates data integration, semantic interoperability, and the construction of unified knowledge bases, which are foundational for numerous applications in real-life scenarios.

In this project, we explore entity alignment using graph embeddings and large language models (LLMs). Initially, we generate vector representations for each node in the knowledge graph using graph embedding techniques. These embeddings enable us to measure the similarity between nodes based on their cosine similarity scores. Our current approach pairs nodes by identifying the top-1 match for each vertex based on this metric. While effective, this method may overlook nuanced relationships that could improve alignment accuracy.

To address this limitation, we propose an enhanced two-stage approach:

1. **Candidate Generation**: For each node, we will identify the top-10 most similar nodes based on cosine similarity scores derived from their graph embeddings.
2. **Refined Pairing**: Using large language models, we will further analyze the string attributes of the candidate nodes to determine the optimal alignment. This step leverages the semantic understanding capabilities of LLMs to refine alignment decisions beyond what numerical similarity scores can achieve.

This hybrid method aims to improve alignment precision by combining the structural insights from graph embeddings with the contextual analysis power of LLMs. By integrating these techniques, the project seeks to advance the state-of-the-art in knowledge graph entity alignment and provide a robust framework for tackling real-world challenges in this domain.

Objectives

The primary objectives of this project are as follows:

1. **Demonstrate the Complementary Benefits of Structural and Entity-Level Information**
   - Show how combining structural embeddings (graph-based information) with entity-specific attributes enhances the accuracy and effectiveness of entity matching in knowledge graphs.
   - Provide empirical evidence supporting the hypothesis that both structural and entity-level information are critical for robust entity alignment.
2. **Showcase the Strength of Combining Basic Embeddings and Large Language Models (LLMs)**
   - Highlight the effectiveness of a two-stage approach where basic graph embeddings generate candidate matches, and LLMs refine these matches using semantic and contextual information from string attributes.
   - Illustrate how integrating these methods bridges the gap between structural similarity and semantic understanding.

Through these objectives, the project aims to advance state-of-the-art techniques for knowledge graph entity alignment and establish a practical framework for leveraging hybrid approaches in real-world applications.

Background and Related Work

Knowledge Graphs and the Entity Alignment Problem

Knowledge graphs (KGs) are powerful data structures used to represent entities and their relationships in a graph format. They are extensively employed in applications such as search engines, recommendation systems, and question answering. However, integrating multiple knowledge graphs is often challenging due to differences in schema, incomplete data, and inconsistencies across graphs.

Entity alignment, a critical step in integrating knowledge graphs, involves identifying equivalent entities across different graphs. This task is complicated by the need to resolve structural, semantic, and syntactic variations.

Graph Embeddings for Entity Matching

Graph embeddings are a popular technique for representing the structural properties of knowledge graphs in a continuous vector space. Methods like DeepWalk, Node2Vec, and Graph Neural Networks (GNNs) encode nodes while preserving their relational and contextual properties. These embeddings are particularly effective for tasks like link prediction, node classification, and entity alignment. However, structural embeddings often overlook the semantic nuances of entities, limiting their alignment performance in complex scenarios.

The Role of Large Language Models (LLMs) in Semantic Analysis

Large Language Models (LLMs) such as GPT and BERT have demonstrated remarkable capabilities in understanding and generating natural language. By leveraging pre-trained contextual embeddings, LLMs can analyze and compare string attributes of entities with a deep semantic understanding. Recent research highlights their potential in augmenting graph-based methods, particularly for tasks requiring fine-grained semantic matching.

Hybrid Approaches for Enhanced Entity Alignment

State-of-the-art approaches for entity alignment often integrate structural and semantic methods to overcome the limitations of individual techniques. For instance, hybrid methods leverage graph embeddings for candidate generation while using LLMs or other semantic models for refining alignment decisions. Despite their promise, these methods face challenges in effectively combining structural and entity-level information, which this project aims to address.

By building on these foundational methods, this project introduces a two-stage approach that combines basic graph embeddings for candidate generation with LLMs for refined matching. This hybrid framework seeks to demonstrate the complementary benefits of structural and semantic information for robust and scalable entity alignment.

Methodology

Dataset

For this project, we chose the Ontology Alignment Evaluation Initiative dataset, which is available here: LINK

Source	Source URL	Language	Hub	Topic	#Instances	#Properties	#Classes	Dump
Star Wars Wiki	http://starwars.wikia.com	en	Movies	Entertainment	145,033	700	269	rdf/xml
The Old Republic Wiki	http://swtor.wikia.com	en	Games	Gaming	4,180	368	101	rdf/xml
Star Wars Galaxies Wiki	http://swg.wikia.com	en	Games	Gaming	9,634	148	67	rdf/xml
Marvel Database	http://marvel.wikia.com	en	Comics	Comics	210,996	139	186	rdf/xml
Marvel Cinematic Universe Wiki	http://marvelcinematicuniverse.wikia.com	en	Movies	Entertainment	17,187	147	55	rdf/xml
Memory Alpha	http://memory-alpha.wikia.com	en	TV	Entertainment	45,828	325	181	rdf/xml
Star Trek Expanded Universe	http://stexpanded.wikia.com	en	TV	Entertainment	13,426	202	283	rdf/xml
Memory Beta	http://memory-beta.wikia.com	en	Books	Entertainment	51,323	423	240	rdf/xml

Fig. 1 Dataset used in the project

This dataset contains scraped Wiki Fandom pages in the form of knowledge graphs. There are 3 different topics (Star Wars, Marvel and Star Trek) and there are 5 given knowledge graph pairs that has to be tested:

- starwars-swg
- starwars-swtor
- marvelcinematicuniverse-marvel
- memoryalpha-memorybeta
- memoryalpha-stexpanded

For example, if we take a look at 'starwars-swg', the task is to pair nodes in the 'starwars' and 'swg' knowledge graphs. The gold node pairs for each graph pair are also given on the website. This way different solutions can easily be evaluated and compared to each other.

Proposed Solution

As the initial step, we clean the input knowledge graphs. The dataset is very unorganized, there are a lot of unnecessary nodes and edges (images, file paths, etc). This way, around half of the nodes and edges were discarded in each network.

After the cleaning step, we create so-called 'dogtags' for each node in the graphs. A dogtag is a string containing the most important information for a given node, namely its label, alternative label, type, abstract and comment.

For example:

```
Label: Gordon Connors
Alternative Label: Gordon Connors
Type: character
Abstract: Senior Lieutenant Gordon Connors (call sign Gordo) was a ...
Comment: His pilot was Colleen Norris.
```

Then we take these dogtags for each node (concatenate all 5 fields into a single string), and calculate embeddings for them. For this task, we use BGE, specifically bge-large-en-v1.5. With this tool, we calculate embedding vectors of 1024 dimensions. The model can be found at Hugging Face here: LINK

Now that we have embedding vectors for every node, we take a graph pair A-B. We take a certain node from graph A and we calculate its cosine similarity to every node in graph B. Our baseline solution is to take the most similar node from graph B (the top-1 solution).

AI Model Integration

During experimetning, we noticed that if we take the top-10 most similar nodes to a selected node, a lot more gold pairs appear. Figure 2 shows this experiment for two graph pairs. On the X axis, -1 means that the desired gold pair is not in the top-20, 0 means the gold pair is in the top-1, …, 19 means the gold pair is in the top-20.

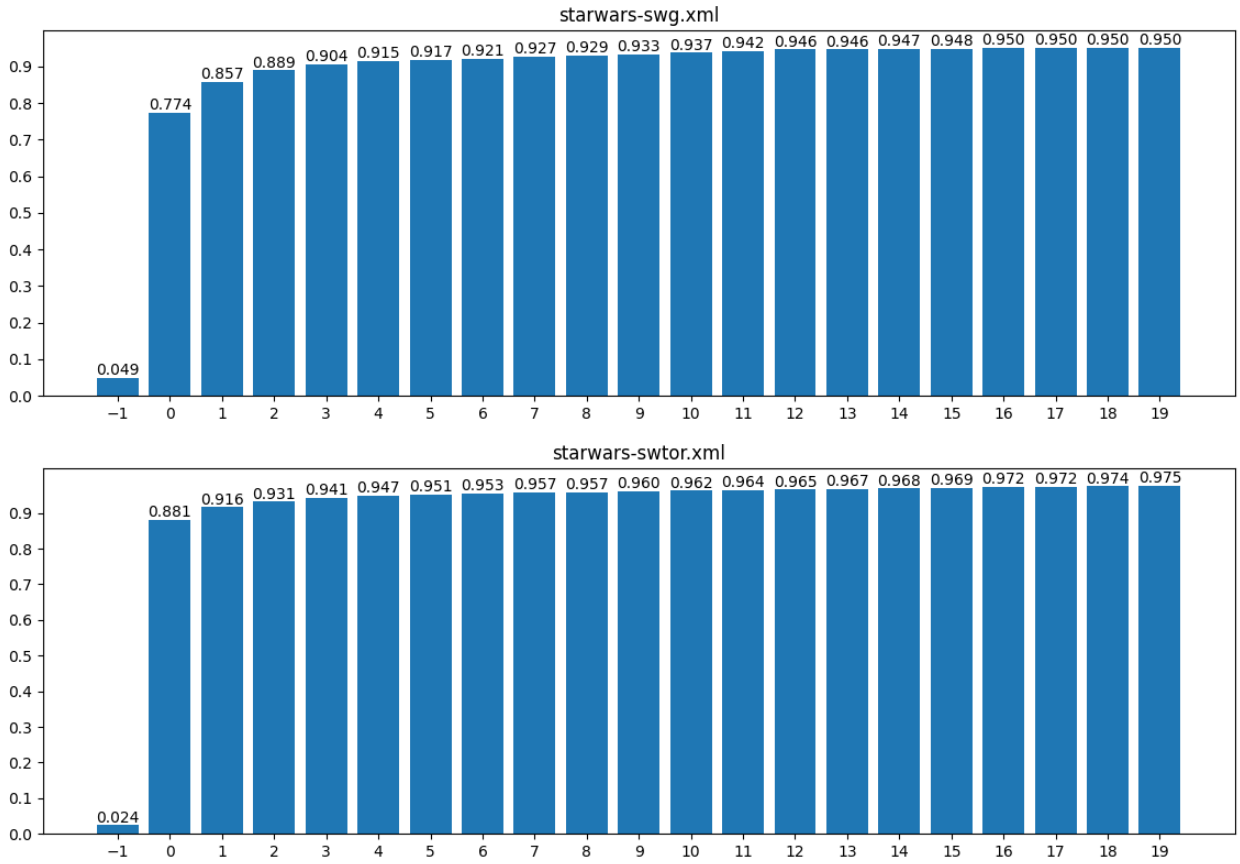

Fig. 2 How many gold pairs are in the top-1 to top-20 most similar nodes?

In other words, if we had a method that can effectively select 1 node from the 10 given options, we could achive better results. Our idea was to integrate AI to this step. We wanted to use an LLM, provide possible solutions and let the LLM choose the node which it thinks is the actual pair. For this purpose, we used OpenAI's 'gpt-4o-mini' model.

Prompt Engineering

We call the OpenAI API with the following prompt:

```
Task: You will be given a description of an anchor entity and a list of
candidate entities, all formatted with XML tags. Your task is to:
- Identify the candidate entity that matches the anchor entity.
- Return the ID number of the matching candidate entity.
- If none of the candidates match the anchor entity, return -1.
###
Anchor: {anchor}
###
{candidate_str}
###
Answer:
```

The 'anchor' is the entity we want to pair, and the candidates are the top-10 most similar nodes from the other graphs. The anchor and the answer together forms a pair.

Implementation Details

The whole codebase was written in Python, using certain packages for different tasks. They are either normal '.py' Python files (for longer runs) or '.ipynb' Jupyter Notebooks (mainly for experimenting and visualizations).

Source code and additional materials can be found in the following GitHub repository: LINK

Technologies Used

The following tools, frameworks and libraries were utilized in the project:

- xml: Parsing the input database to a more human-readable format.
- json: Store and read saved data easily.
- torch: For machine learning tasks.
- sentence_transformers: Embedding dogtags with BGE.
- fuzzywuzzy: Calculate string similarity for deduplication.
- tqdm: Measuring runtime of certain code parts.
- pandas: Working with large amounts of data.
- numpy: Working with large amounts of data.
- matplotlib: Creating charts for visualization.
- openai: For OpenAI API calls.
- networkx: Working with graphs.

Code Organization

A short description of each code file from the GitHub repository:

- alignment_format: Helper file to read the 'rdf' formatted input data.
- cosinesimilarity_top10pairs: For a node in graph A, calculate the cosine similarity compared to every other node in graph B. Keep the top-10 most similar nodes and repeat the process for each node in graph A.
- deduplicate_fuzzy: If multiple exact matches are found, keep the one that has the most similar URL.
- dogtag_embeddings: Create dogtags for each node and calculate their sentence embeddings using BGE.
- edge_browser: Helper file to browse the original input networks.
- eval: Helper file for evaluation purposes.
- evaluator: For calculating our final results (precision, recall and F1-score for each setting).
- exact_matching: Find exact matches in node labels.
- gold_top10_overlap: Checking the percentage of gold pairs found in the top-1 to top-20 range.
- gold_top10pairs_recall: Checking the percentage of gold pairs found in the top-10 to top-1000 range.
- gptcall_top10: Calling the OpenAI API to select one node from the top-10 possible options.
- intellisense: Helper functions for calculating recall.
- loaders: Helper file to load data.

Results

- Summary of findings or outcomes, with data visualizations or examples if applicable.

Discussion

Impact on Research Area

Our project contributes to the field in many aspects. First of all, we couldn't find a lot of solutions in the literature for the entity alignment problem. Secondly, to our knowledge, the proposed solution is one of the first algorithms to use graph embeddings in combination with LLMs to provide a solution. We also have to highlight that with our methodology, a lot of LLM resources can be saved, since the search space of the alignment problem is quite small. This ensures that the algorithm can be applied to large networks even with millions of nodes

and edges, without exceeding the context window of the LLM used.

Limitations and Future Work

We acknowledge that our methodology is not perfect and has some drawbacks. We could have tried multiple other promtps to find out which achieve the best performance. Cleaning the unorganized date as the initial step was also quite hard, and maybe we deleted meaningful information during the process that could later be useful for the entity alignment task.

Regarding future work, we could use bigger sentence embedding models or better LLMs, but this is just an initial step from us for researching the field and our budget is very limited. However, we can agree that despite all the limitations, we could still achieve competitive results for the chosen dataset.

Conclusion

In conclusion, [TODO]

Appendices

1. GitHub Repository with source code and additional materials: LINK
2. Dataset: LINK
3. BGE string embedding: LINK