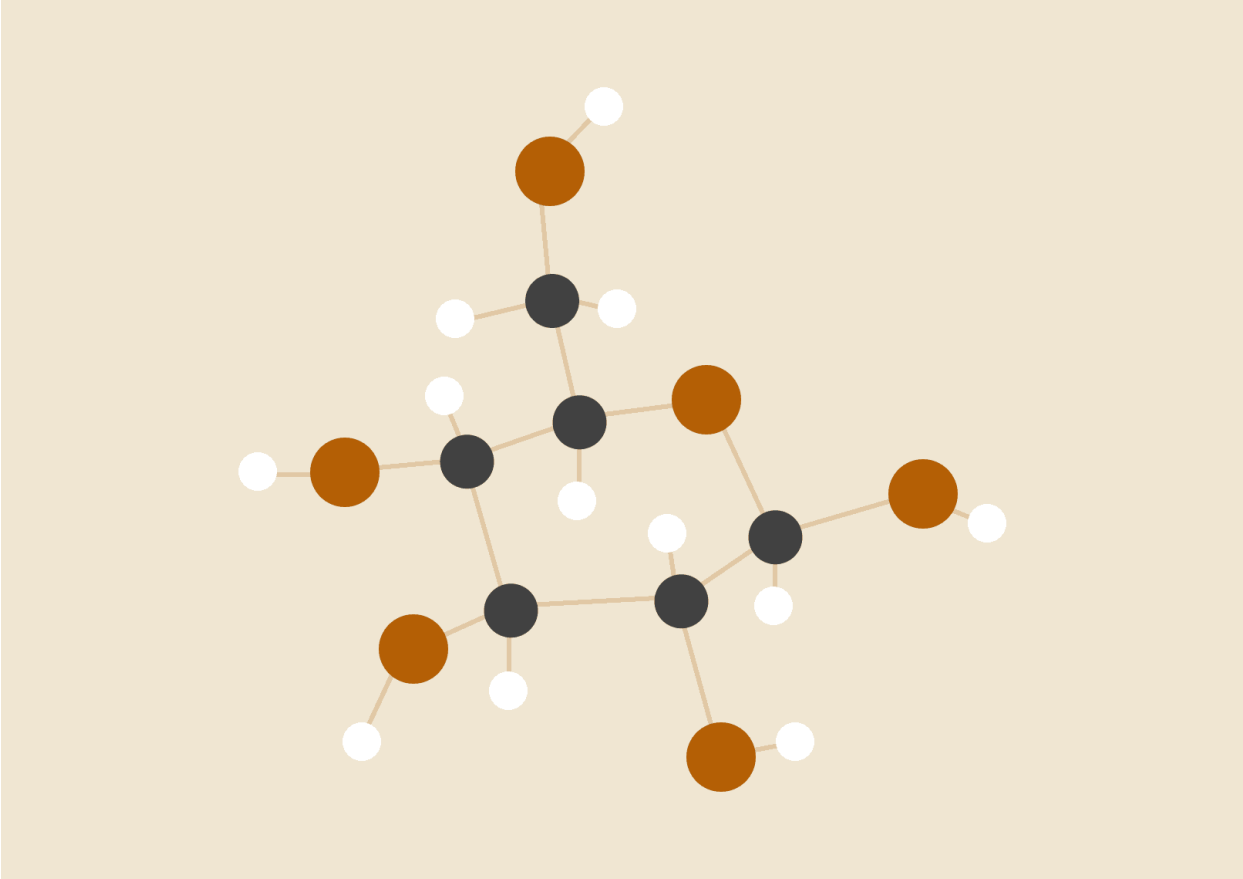
Ontology Alignment Using Lexical Matching



Nima Sahraneshinsamani

Jun 2024

This report has been generated with the assistance of GPT. This includes the text you are currently reading, as well as all related code and programs mentioned herein.

INTRODUCTION

Ontology alignment involves comparing and establishing relationships between entities from different ontologies to enable interoperability and information exchange. This report focuses on designing and implementing a basic ontology alignment system, specifically employing a lexical matching approach.

Task B.1: Basic Ontology Alignment

Basic ontology alignment simplifies to comparing lists of elements from two ontologies.

Task B.2: Design and Implementation of Lexical Matcher

The lexical matcher is designed and implemented to load two ontologies and find equivalence correspondences among their entities.

Task B.3: Apply Algorithm to OAEI Conference Track Ontologies

The implemented algorithm is applied to the following pairs of ontologies from the OAEI's conference track:

- cmt.owl - ekaw.owl
- cmt.owl - confOf.owl
- confOf.owl - ekaw.owl

Objective: Align ontologies from the OAEI conference track (`cmt.owl`, `ekaw.owl`, `confOf.owl`) using lexical matching techniques and evaluate the mappings.

Steps Taken:

1. **Loading Ontologies:**
    - We used the `Owlready2` library to load the ontologies `cmt.owl`, `ekaw.owl`, and `confOf.owl`.
2. **Lexical Matching:**
    - Implemented a lexical matcher that compares classes between pairs of ontologies.
    - Used lexical similarity metrics (`Jaro-Winkler` and `ISub`) to determine similarity between entity labels.
3. **Alignment Generation:**
    - For each pair of ontologies (e.g., `cmt.owl` and `ekaw.owl`), generated equivalence mappings based on lexical similarity above a specified threshold (`0.8`).
4. **Output:**
    - Saved the generated mappings as RDF triples (`turtle` format) in output files (`cmt-ekaw-matches.ttl`, `cmt-confof-matches.ttl`, `confof-ekaw-matches.ttl`).

Task B.4: Apply Algorithm to OAEI Anatomy Track Ontologies

Objective: Extend the alignment algorithm to larger ontologies (`mouse.owl` and `human.owl`) from the OAEI anatomy track, and evaluate the mappings.

Steps Taken:

1. **Loading Ontologies:**
    - **Loaded the anatomy track ontologies `mouse.owl` and `human.owl` using `Owlready2`.**
2. **Lexical Matching:**
    - **Reused the lexical matching function to compare classes between `mouse.owl` and `human.owl`.**
3. **Alignment Generation:**
    - **Applied lexical similarity metrics to identify equivalent classes between the large-scale ontologies.**
4. **Output:**

    - Saved the generated mappings as RDF triples (`turtle` format) in an output file (`mouse-human-matches.ttl`).
5. Evaluation:
    - Compared the generated mappings against a reference alignment (`anatomy-reference-mappings.ttl`).
    - Computed precision, recall, and F-score to assess the quality of the mappings.onn

In this task, I have a sequential version and two parallel versions of the code also to perform the task more efficiently.

- B4_sequential.py
- B4_parallel_nested.py
- B4_parallel_chunk_based.py
-

The explanation of each parallel method and the results of the execution are as follows:

Parallelization Strategies

Two parallelization strategies were implemented to optimize the computation-intensive task of comparing entity labels across large ontologies:

1. Nested Loop Parallelization:
    - The comparison of entity labels was parallelized by distributing the nested loops across multiple processes using Python's multiprocessing library. This approach aimed to exploit multi-core processors effectively.
    - Evaluation metrics were computed to assess the quality of mappings produced.
2. Chunk-based Parallelization:
    - Entities were split into chunks, and each chunk was processed in parallel. This method allowed for more granular control over parallel execution and potentially reduced overhead compared to the nested loop approach.

- Similar evaluation metrics were generated to compare results with the sequential and nested loop parallelization methods.

Results and Discussion

Performance Comparison

- **Sequential Time: 1006.96 seconds**
- **Nested Loop Parallel Time: 189.26 seconds**
- **Chunk-based Parallel Time: 356.92 seconds**

- Evaluation Metrics:

Method 1: Nested Loop Parallelization

- **Precision: 0.5974**
- **Recall: 0.6313**
- **F-Score: 0.6139**

Method 2: Chunk-based Parallelization

- **Precision: 0.5974**
- **Recall: 0.6313**
- **F-Score: 0.6139**

Observations:

1. **Time Comparison:**
   - **Method 1 (nested loop parallelization) achieved a significant reduction in execution time compared to the sequential approach (from 1006.96 seconds to 189.26 seconds).**
   - **Method 2 (chunk-based parallelization) also reduced execution time but not as significantly as Method 1, suggesting that further optimization or tuning may be possible.**
2. **Evaluation Metrics:**
   - **Precision, recall, and F-score values are identical across both**

parallelization methods and match the sequential approach. This indicates that the mappings generated are consistent and accurate regardless of the parallelization strategy used.