Ontology alignment for Linked Data

MASTER 2 DATA & KNOWLEDGE

Information Integration

Presented by :

Wafa DJERAD

Wilfried ZAKIE

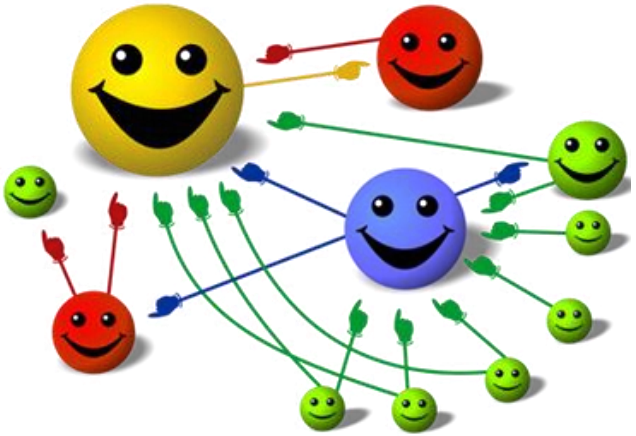
Plan

Introduction

I. **Matching Approach**
II. **Experimental setting**
III. **Results**

Conclusion

Introduction



Data linking

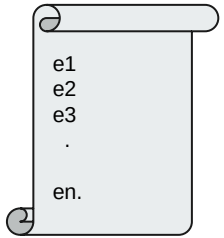
- ➔ Bring together information from different sources
- ➔ Create a new and richer dataset

Combine records from different datasets if they refer to the same entity or a very similar one.

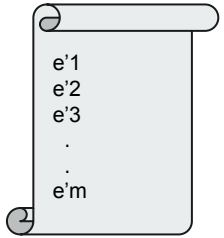
Data Linking

Given two ontologies A and B, an alignment between A and B is a set of correspondences :

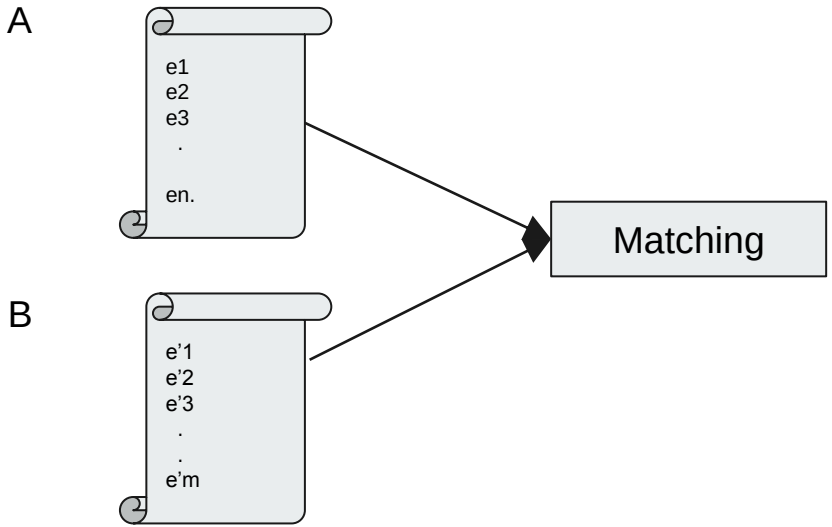
A



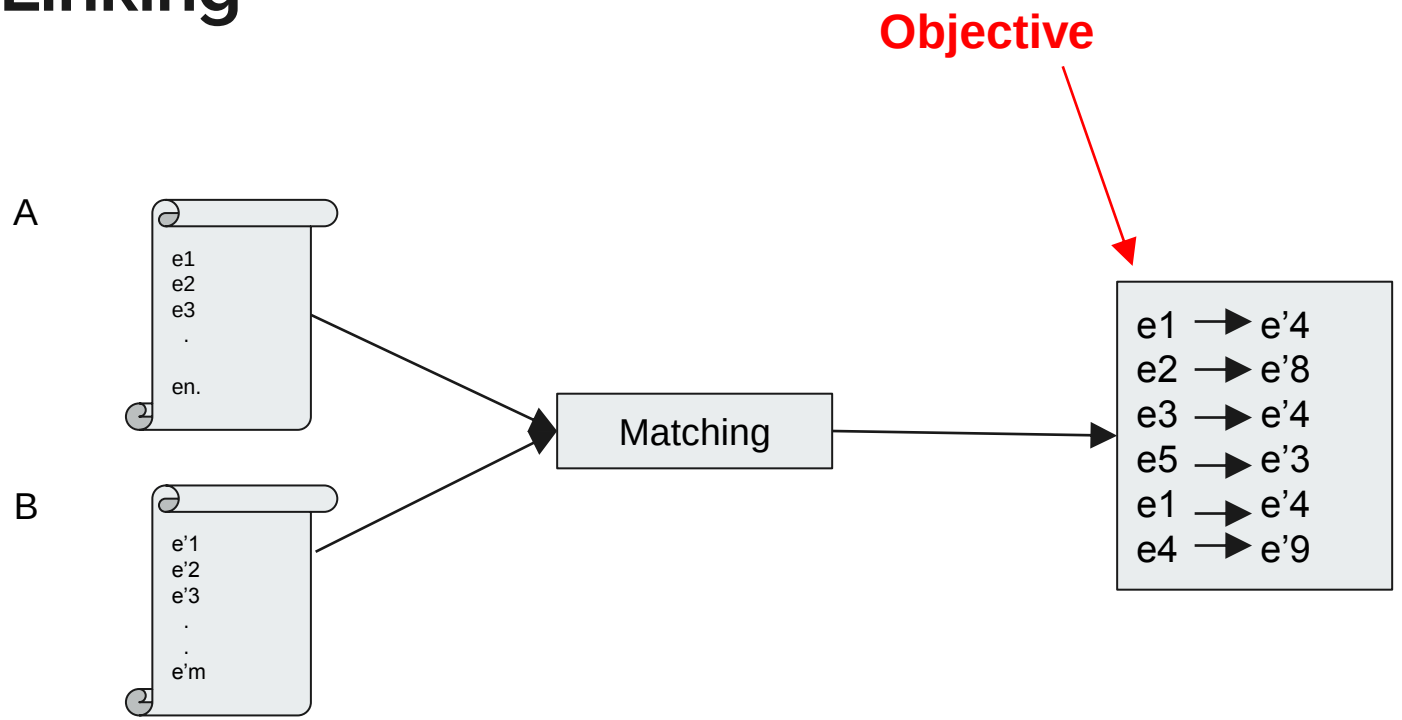
B



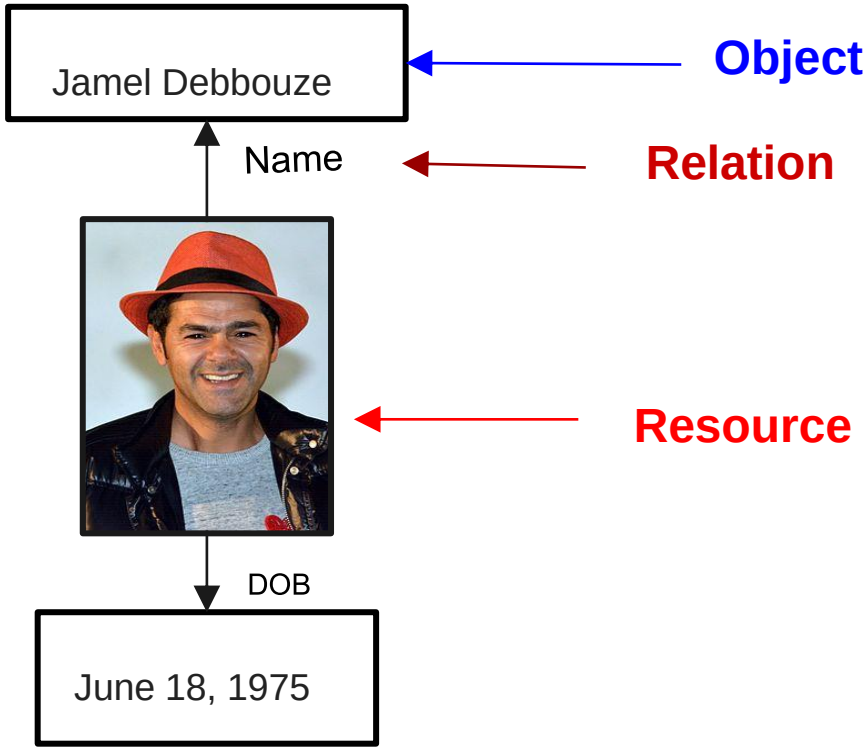
Data Linking



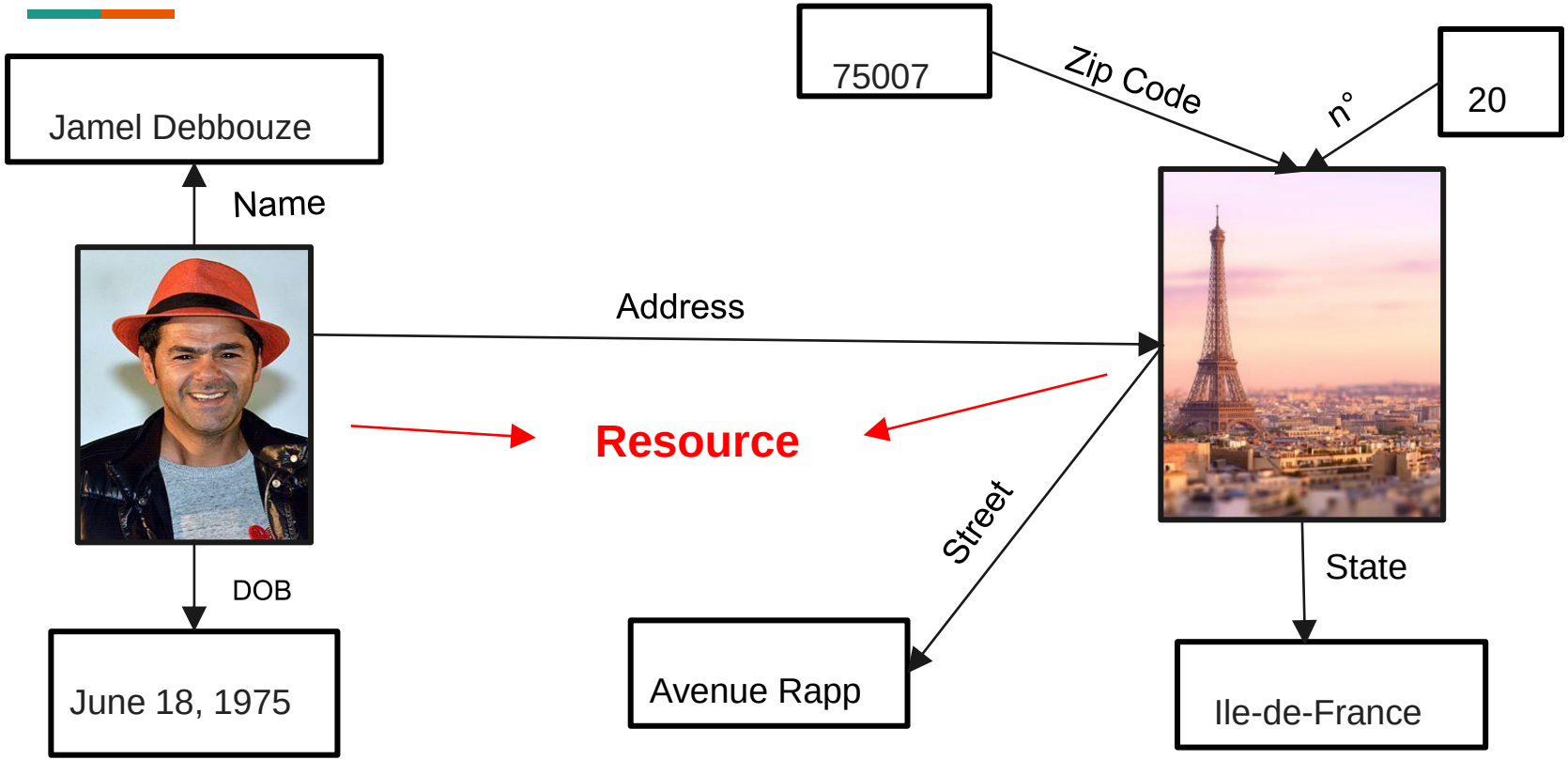
Data Linking



RDF graph for Ontology Representation

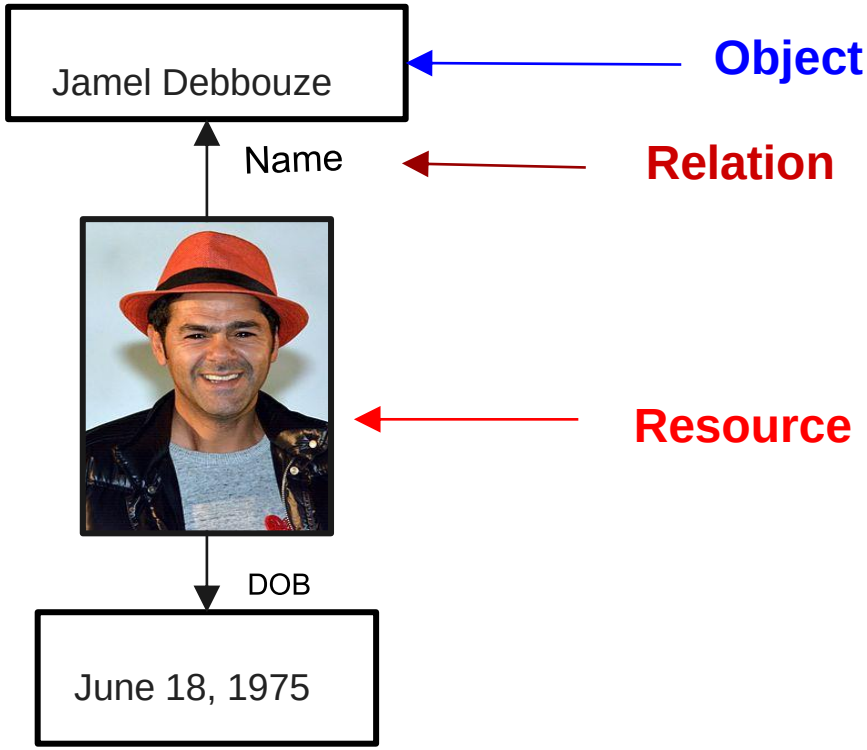


Mapping Approach



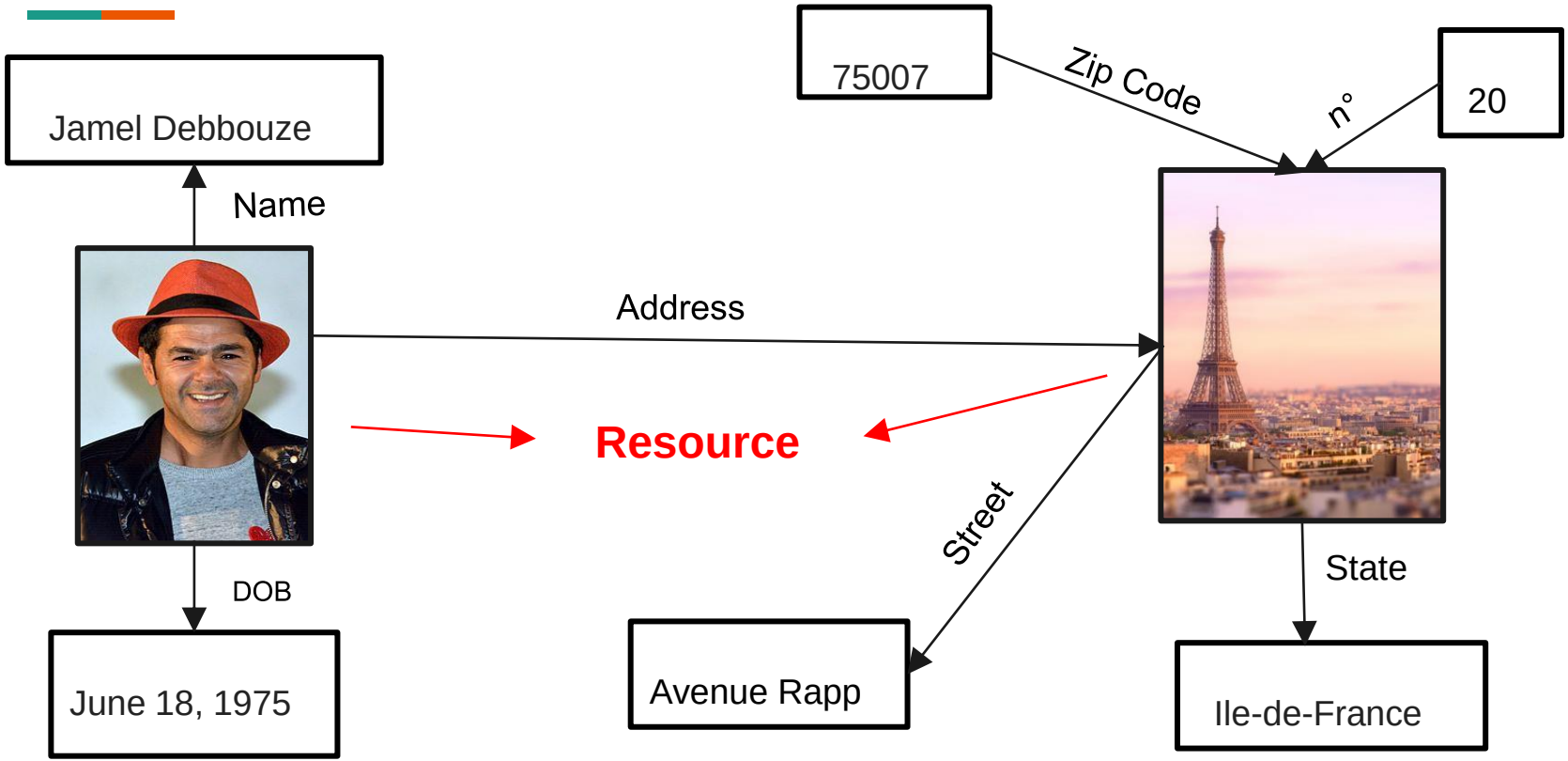
RDF graph query

Select * where ?s ?r ?o1

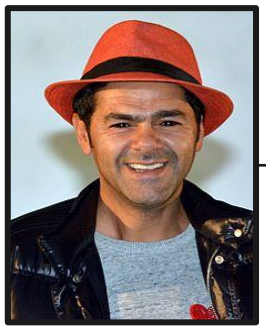


Mapping Approach

Select * where ?o1 ?r ?o



Map resource to all linked objects



Jamel Debbouze

20

Avenue Rapp

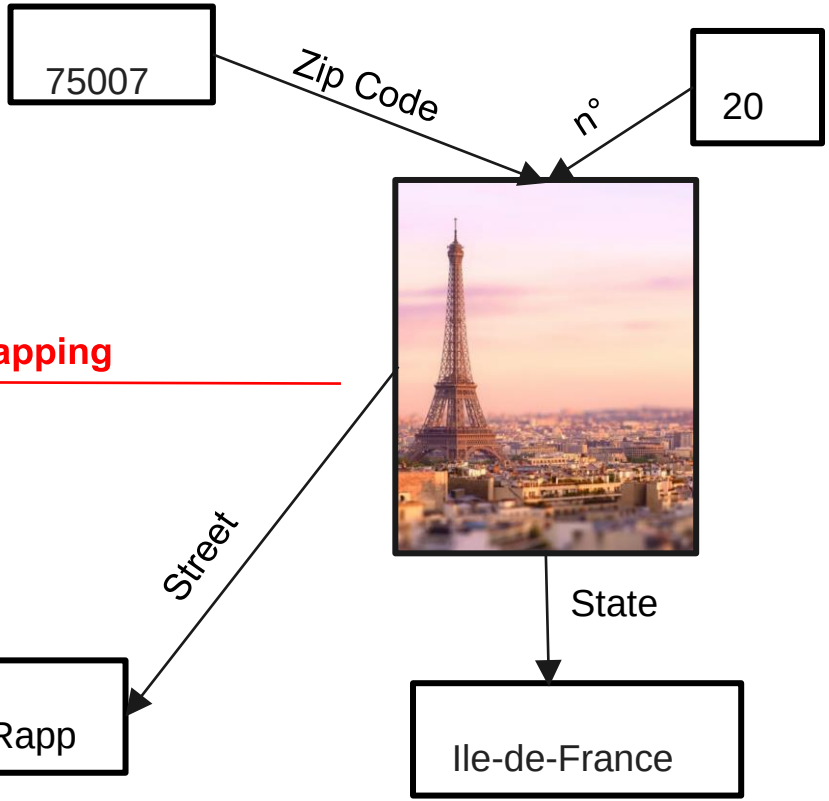
75007

Ile-de-France

June 18, 1975

Mapping Approach

Will not be used for mapping



Mapping Approach

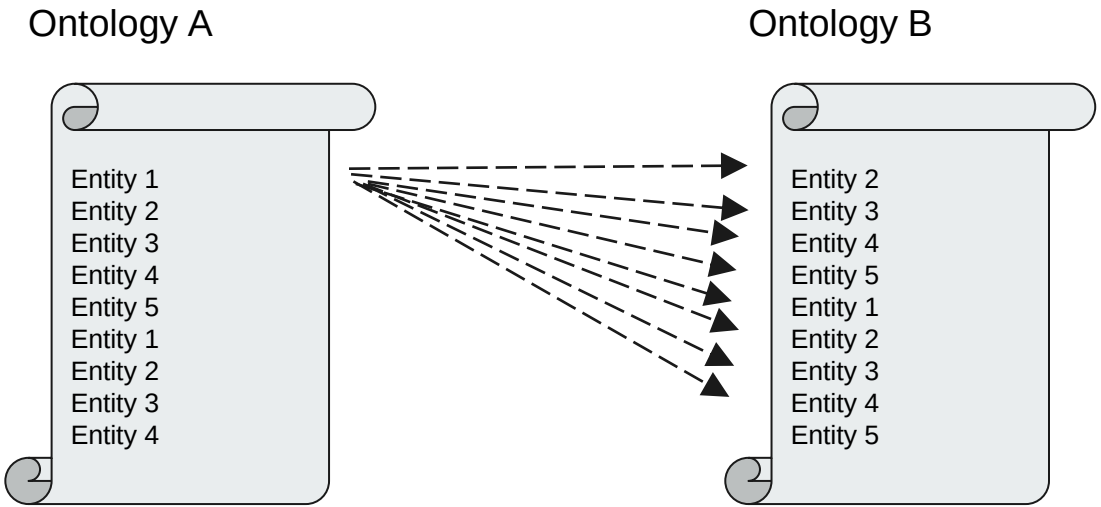
SCORE

- ➔ If the string of two literals (objects) are similar give a weight of 1
- ➔ Otherwise compute the similarity between them using string similarity methods

The entity having the highest score is chosen as the match

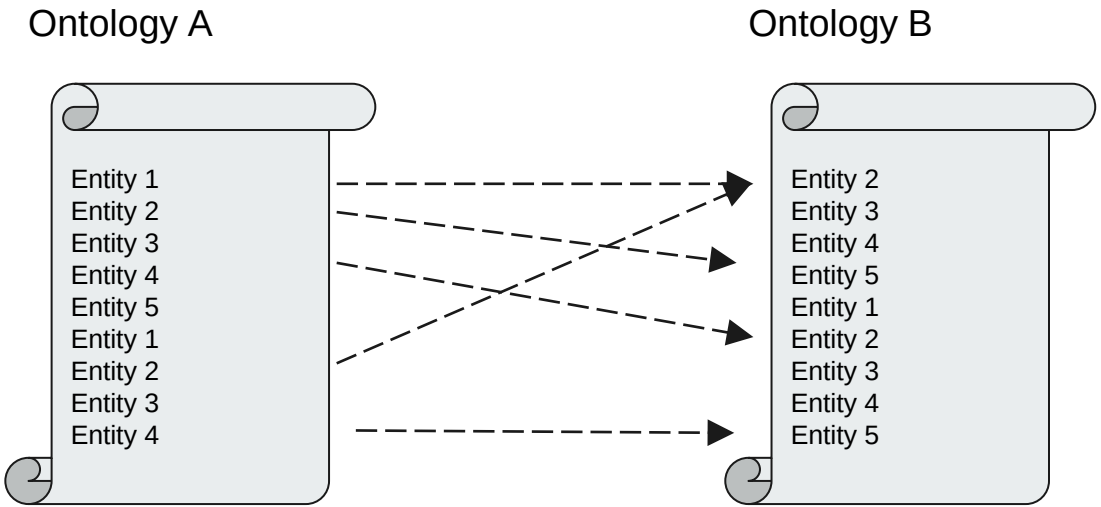
We process the following steps

Mapping Approach



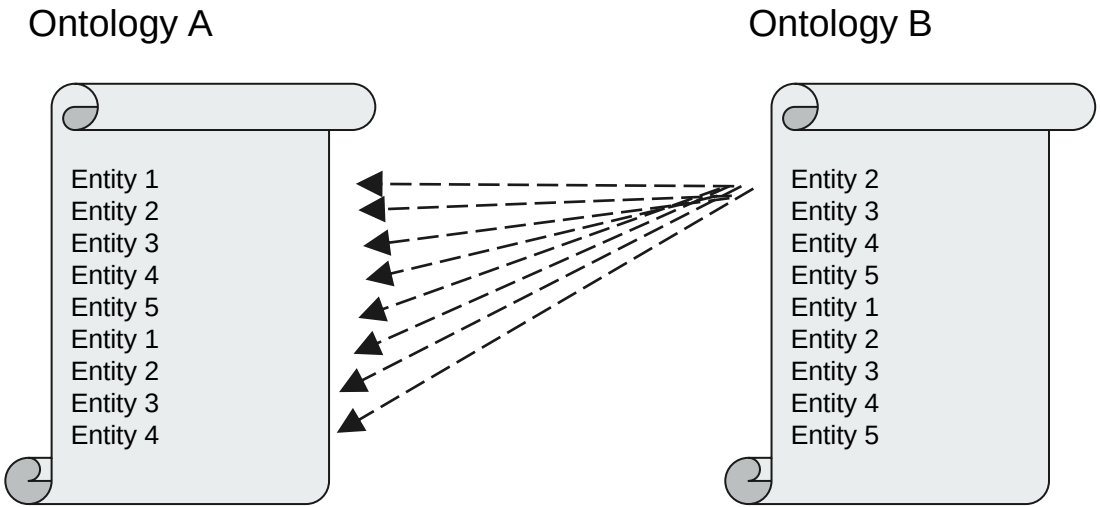
Compute similarity score between objects of each entity from ontology A and objects of all entity in ontology B

Mapping Approach



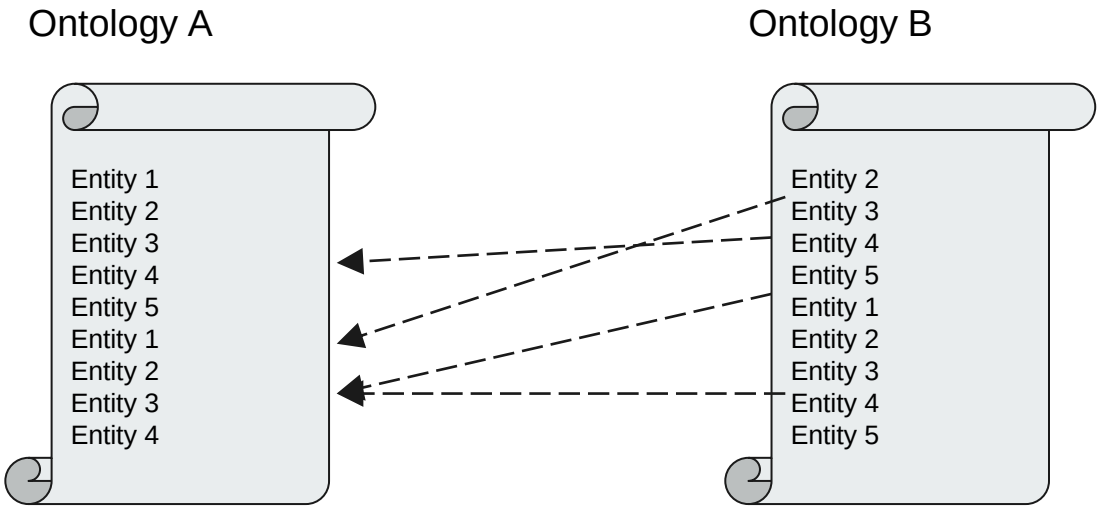
Result : n to 1 mapping

Mapping Approach



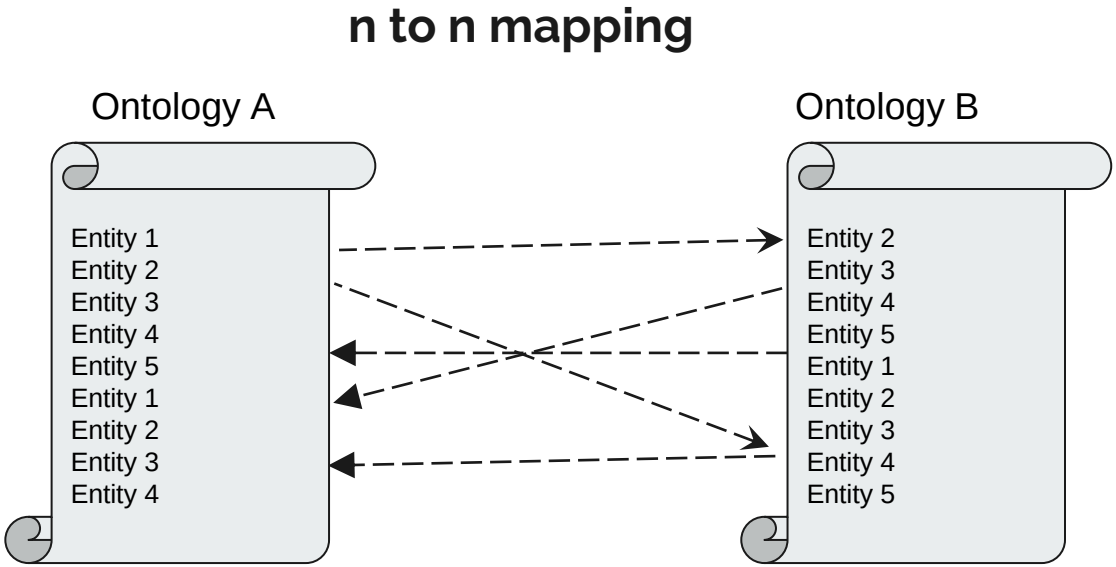
Compute similarity score between objects of each entity from ontology B and objects of all entity in ontology A

Mapping Approach



Result : 1 to n mapping

Mapping Approach



Results= union of both methods

Evaluation

We compute the precision and the recall for the result with respect to the Gold Standard.

$$\text{Precision} = \frac{\text{number Correct}}{\text{size of Output file}} \qquad \text{Recall} = \frac{\text{number Correct}}{\text{sizeOfGoldStandard}}$$

Experiment Setting

3 Tested DataSet + GoldStandards:

- Person 1
- Person 2
- Restaurant

API:

- Jena
- SecondString

Similarity Measure

Jaro Winkler

- A string metric for measuring the edit distance between strings
- An extension of Jaro by considering the size of the longest prefix between S1 and S

Jaro_Winkler (S1, S2) = Jaro (S1, S2) + L * p * (1 - Jaro (S1, S2))

Where:

- L is the length of common prefix at the beginning of the string up to 4.
- p is a scaling factor not exceed 1/4.

Similarity Measure

Jaro Winkler

- A string metric for measuring the edit distance between strings
- An extension of Jaro by considering the size of the longest prefix between S1 and S2

$$\text{Jaro\_Winkler}(S1, S2) = \text{Jaro}(S1, S2) + L * p * (1 - \text{Jaro}(S1, S2))$$

Where:

- L is the length of common prefix at the beginning of the string up to 4.
- p is a scaling factor not exceed 1/4.

Jaccard

$$\text{Jaro\_Winkler}(S1, S2) = \frac{\text{Intersection}(S1,S2)}{\text{Union}(S1,S2)}$$

Similarity Measure

Computing the similarity score between two objects is equivalent to:

- Give score of 1 if both string are equals
- Compute the mentionned string similarity

Score = (JaroWinkler (o, o') + Jaccard(o, o')*2)/3

Similarity Measure

The aggregate score of all objects for two given entities is normalized

$$\textbf{Normalized\_Score} = \frac{\sum \textbf{scores}}{\textbf{size of objects}}$$

We will only store as matching entities the pair of entities having the greatest score among the one whose score is > threshold (0.8)

Results

DataSets/Metrics	Person 1	Person 2	Restaurant
Precision	99.60 %	82.21 %	77.87%
Recall	100.0 %	76.25 %	77.57 %

Conclusion

- We computed an algorithm for data linking based on string similarity
- In our approach we compared entities by using the objects related to it
- As result we keep entity’s pairs having the best aggregate scores
- This algorithm has been tested on 3 different datasets
- Discover relevant common keys to efficiently query the RDF graphs using SAKey

Thanks !

To improve our results

Need of normalization when comparing entities

- Use normalization methods for data property (attribute) values:
    - lemmatization
    - Stop words remove
    - Enforce common abbreviations
    - Part of ETL tools, commonly using field segmentation and dictionaries

- **Token based (e.g. Jaccard, TF/IDF cosinus) :**

  The similarty depends on the set of tokens that appear in both S and T.

- **Edit based (e.g. Levenstein, Jaro, Jaro-Winkler) :**

  The similarty depends on the smallest sequence of edit operations which transform S into T.

- **Hybrids (e.g. N-Grams, Jaro-Winkler/TF-IDF, Soundex)**

- **Jaccard measure:** **Jaccard(S,T) = |S∩T| / |S∪T|**

  Jaccard(« rue de la vieille pierre », « 11 rue vieille pierre ») =3/6

- **Cosinus (based on TF-IDF)**

  Widely used in traditional information retrieval (IR) approaches

  - Intuition: a term that is rare in the data is important and a term that is frequent in the string (value) is important.
  - Term frequency(TF): # of times a 'term' appears in the string compared with the size of the string.
  - Document frequency (IDF): the inverse of (# strings that contain the 'term'/ # of strings in the corpus)

Cosinus computation based on TF-IDF

- Compute for each value the set of terms represented in a vector of terms
- Compute for each term its weight TF-IDF:

$$V(w,S) = V'(w,S) / \sqrt{\sum_{w'} V'(w',S)^2}$$

With $V'(w, S) = \log(TF_{w,S} + 1).\log(IDFw)$

Let s, t be two values, S, T the sets of terms resp. and
V(w, S), V(w, T) the weights of the term w in S and T, resp.

$$Cosinus(s,t) = \sum_{w \in S \cap T} V(w,S) * V(w,T)$$

Example :

Low weights for "Corporation", high weights for "AT&T", "IBM"

Cosinus("AT&T", "AT&T Corporation") high

Cosinus("AT&T Corporation", "IBM Corporation") Low

- **Jaro**
  - For (S, T), the character *c* is common for (S, T):
    if (S_i=c), (T_j=c), and |i-j| < min(|S|,|T|) / 2.
  - The character *c and d* are ***transpositions*** if *c and d* are common for S and T and appear in different orders in S and T.

$$Jaro(S,T) = \frac{1}{3}\left(\frac{m}{|S|} + \frac{m}{|T|} + \frac{m-t}{m}\right)$$

  - Example: Jaro(Texas, Texhas) $= \frac{1}{3}\left(\frac{5}{5} + \frac{5}{6} + \frac{5-2}{5}\right) = 0{,}81$

Jaro-Winkler

- An extension of Jaro by considering the size of the longest prefix between S and T.

$$Jaro-Winkler(S,T) = Jaro(S,T) + \left(\frac{\max(P,4)}{10} * (1 - Jaro(S,T)) \right)$$

- Example : Jaro-Winkler(Texas, Texhas) $= 0{,}81 + \left(\frac{4}{10} * (1 - 0{,}81) \right)$

$= 0{,}88$

- Runtime efficiency
- Showed to be relevant for the comparison of person names [Cohen03].