Index-based semantic tagging for efficient query interpretation

José Devezas & Sérgio Nunes

INESC TEC & DEI, Faculdade de Engenharia, Universidade do Porto

Universidade do Porto
Faculdade de Engenharia
FEUP

Entity-Oriented Search

- Modern search engines are evolving beyond ad hoc document retrieval.
- Nowadays, the information needs of the users can be directly satisfied through entity-oriented search:
  - By ranking the entities or attributes that better relate to the query;
  - As opposed to the documents that contain the best matching terms.
- One of the challenges in entity-oriented search is efficient query interpretation.
- The task of semantic tagging is central to understanding user intent.

Semantic Tagging in Queries

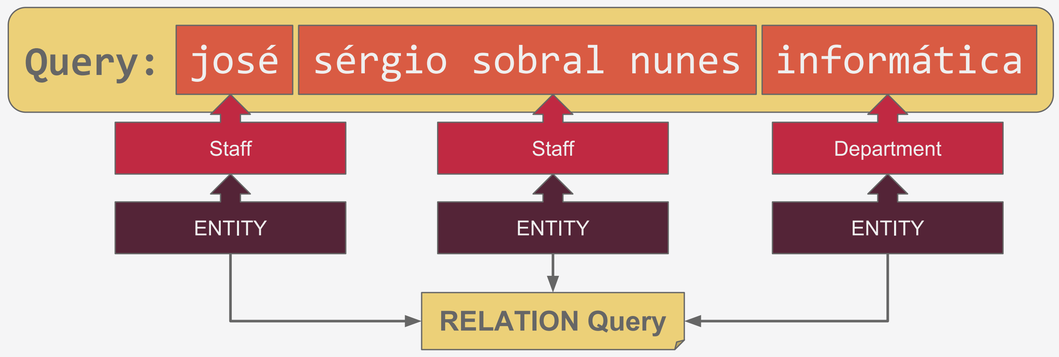


Figure 1: Segmented query, annotated with the most probable entity type and category for each query part, as well as the overall query category.

Five Query Categories (Pound et al., 2010)

1. Entity Query
   - Directly find a specific entity.
2. Type Query
   - Find entities of a given type.
3. Attribute Query
   - Find values for an attribute of an entity or type.
4. Relation Query
   - Discover how two or more entities or types are connected.
5. Keyword Query
   - For any traditional full-text query that doesn't fit the other categories.

Use Case: Entity-Oriented Search at the University of Porto

- Main goal: improving search at the University of Porto by taking advantage of the untapped underlying linked data present in the current information system.
- Some of the search tasks included:
  - The discovery of the department for a particular staff member;
  - Finding students enrolled in a specified group of courses.
- We first tackled this problem at a faculty level.
- And then extended our support to the fourteen schools of the University of Porto.
- Our initial semantic tagging implementation was directly supported on SPARQL queries over the Sesame triple store.
- This led to performance issues when scaling from faculty-centric entities to university-centric entities.
- The triple store grew from 546,760 to 2,594,511 statements.
- This translated into:
  - 139,640 more students, associated with 193,650 additional enrollments;
  - 1,166 more courses;
  - 14 more academic years;
  - 10 more faculties.

Probabilistic Semantic Tagging in Queries

- Semantic tagging in queries is the act of annotating queries with entity types.
- It is essential for query interpretation and understanding.
- We segmented the query and annotated groups of sequential terms (n-grams) with the most probable category (*entity*, *attribute*, *type* or *keyword*), based on a set of matching candidate labels from the knowledge base.
- We focused on the efficiency of two alternative methodologies for candidate retrieval:
  - One based on a Sesame triple store and SPARQL querying;
  - And another one based on a Lucene index and keyword querying.
- The first step for query analysis was to build a collection of all n-grams for $n \in [1, n]$ (Figure 2).
- We used $n = 6$ as the maximum n-gram size, given it provided a coverage of 94.28% for the labels of our entities (Table 1).
- This was a good compromise between performance and accuracy.
  - A higher number of n-grams would result in additional candidate retrieval queries.

Table 1: Label term count distribution.

Terms	Freq.	Coverage
22	1	100.0000%
14	1	99.9994%
12	2	99.9989%
11	23	99.9978%
10	93	99.9851%
9	402	99.9339%
8	2173	99.7126%
7	7690	98.5162%
6	**22260**	**94.2823%**
5	48308	82.0265%
4	65598	55.4295%
3	22089	19.3130%
2	12980	7.1514%
1	9	0.0050%

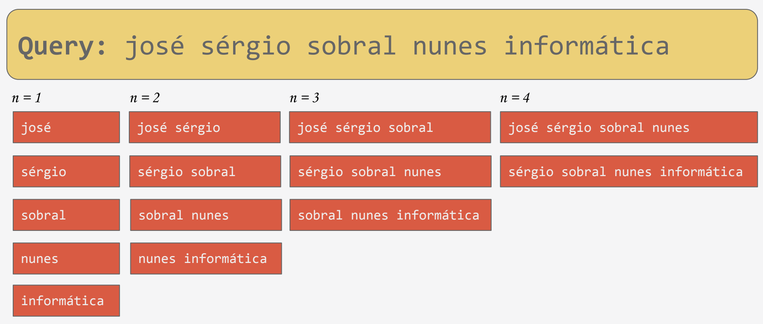


Figure 2: Example collection of n-grams, for a maximum n-gram size of $n = 4$.

- The second step was to retrieve matching candidates for each n-gram.
- We did this with the Sesame triple store and the Lucene index.
- We also computed the number of candidates per class.
- This enabled us to calculate the probability of associating a given candidate to an n-gram:

$$1 - \frac{|C_t^x|}{|C_t|}$$

where C_t^x is the set of candidates for n-gram x and type t, and C_t is the set of candidates for type t.

- This probability is higher when the fraction of candidates is smaller, which means that rarer labels will have priority over common labels, resulting in better precision.
- Finally, in the last step, we selected the n-gram with the highest probability, keeping only the longest n-gram in case of term overlap between selected n-grams.
- Each candidate could be directly categorized into *entity*, *attribute* or *type*, based on its knowledge base class.
- This information was used to classify the query, based on pre-made templates that mapped a set of categories to a query class (e.g. a query with an *entity* and an *attribute* was classified as an ATTRIBUTE Query).

Candidate Retrieval Strategies

- Our first attempt at retrieving matching candidates was directly based on the **Sesame triple store**.
- We first experimented with a knowledge base containing 546,760 statements.
- While this approach did not allow for sub-second query times, it resulted in a reasonable query time of under 5 seconds.
- The SPARQL query we built returned four columns associated with candidate entities: Label, URI, Class and Category. This was obtained from the union of three sub-queries for *entity*, *attribute* and *type* individuals.
- Candidate retrieval was done through filtering, using a case insensitive regular expression that matched the n-grams generated from the search query.
- As an alternative for better performance, we built a **Lucene index** based on the triple store data, combining documents for *entities*, *attributes* and *types*.
- Candidate retrieval was done by querying the index with each n-gram generated from the search query.
- We used proximity search within $n = 6$ terms of distance (the same as the n-gram size) and ensured that the query was parsed in-order.
- For each query, we returned the top-$\mathcal{N}$ results.

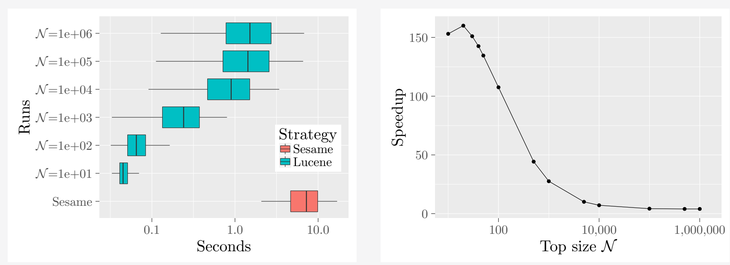
Evaluating Efficiency

- We compared the performance of both candidate retrieval strategies by measuring overall search time over a set of test queries.
- We synthetically generated 1,000 test queries with 3 to 8 terms from random ontology individuals and 0 to 2 terms from a Portuguese dictionary with over 400,000 words.
- We did a run based on the Sesame triple store strategy, that we directly compared with a run based the Lucene index strategy for the top-$\mathcal{N}$ results.
- We picked $\mathcal{N} = 10$ since it provided a near-optimal speedup, also having a positive impact on the quality of the results for a small set of manually tested queries.
- Tests were run on a laptop with a dual core Intel® Core™ i7-5600U, 16 GB of RAM and a 256 GB solid-state drive.
- As seen on Table 2, the Lucene index strategy was nearly 153 times faster than the Sesame triple store strategy, for $\mathcal{N} = 10$.

Table 2: Statistics for the query analysis time of the Sesame triple store and the Lucene index strategies, using $\mathcal{N} = 10$ for the Lucene index.

	Sesame triple store	Lucene index
Avg.	7.435765s	0.048580s
Std.	$\pm 3.206806s$	$\pm 0.019115s$
Speedup	153.062268 ($\sim 153\times$ faster)	
Mann-Whitney U Test	p-value $\approx 0 \ll 0.01$	

- Increasing the parameter $\mathcal{N}$ resulted in lower, but still positive, speedup values.



(a) Run times for the Sesame and Lucene strategies (log scale for the x-axis).

(b) Speedup for different values of $\mathcal{N}$ (log scale for the x-axis).

Figure 3: Efficiency evaluation of the overall search process. The same 1,000 synthetic queries were used in each run.

- Figure 3a shows a run time comparison between the Sesame strategy (all matching results) and various $\mathcal{N}$ values of the Lucene strategy (top-$\mathcal{N}$ results).
- The index-based strategy outperforms the triple store strategy even when retrieving the top $\mathcal{N} = 1$ million matching candidates.
- Figure 3b illustrates the evolution of the speedup for growing values of $\mathcal{N}$.
- For $\mathcal{N} > 20$, the speedup consistently decreased, nearly stabilizing at $4\times$ faster.