Optimizing the Computation of a Possibilistic Heuristic to Test OWL SubClassOf Axioms Against RDF Data

Rémi Felin, Olivier Corby, Catherine Faron, Andrea G. B. Tettamanzi

Optimizing the Computation of a Possibilistic Heuristic to Test OWL SubClassOf Axioms Against RDF Data

Rémi FELIN
Université Côte d'Azur, Inria, I3S
Sophia-Antipolis, France
remi.felin@inria.fr

Olivier CORBY
Université Côte d'Azur, Inria, I3S
Sophia-Antipolis, France
olivier.corby@inria.fr

Catherine FARON
Université Côte d'Azur, Inria, I3S
Sophia-Antipolis, France
catherine.faron@inria.fr

Andrea G. B. TETTAMANZI
Université Côte d'Azur, Inria, I3S
Sophia-Antipolis, France
andrea.tettamanzi@inria.fr

***Abstract*—The growth of the semantic Web requires tools to manage data, make them available to humans and for a wide range of applications. In particular, tools dedicated to ontology management are a keystone for semantic Web applications. In this paper we consider a possibilistic framework and an evolutionary approach for ontology enrichment with OWL axioms. The assessment of candidate OWL axioms against an RDF knowledge graph requires a high computational cost, especially in terms of computation time (CPU), which may limit the applicability of the framework. To answer this problem, our contribution presented in this paper consists of (i) a multi-threading system to parallelize axiom assessment, (ii) a heuristic to avoid redundant computation and (iii) an optimization for SPARQL query chunking relying on an extension of the SPARQL 1.1 Federated Query standard. The results of a comparative evaluation show that our proposal significantly outperforms the original algorithm, enabling a significant reduction in computation time.**

***Index Terms*—Knowledge Graphs, OWL Axioms, Ontology Enrichment**

I. Introduction

Over time, the semantic Web has developed significantly, relying on a series of W3C standards, the foremost of which are RDF (Resource Description Framework), SPARQL (SPARQL Protocol and RDF Query Language) and OWL (Web Ontology Language). As a result, the use cases have diversified widely in the domains of academia, research and industry. The benefits of the semantic Web are diverse and revolve around three axes [1]: **interoperability** between systems, notably through the addition of standards allowing the creation and distribution of knowledge graphs through the Web. These graphs are linked together when they describe the same resources or use the same vocabularies. **Linked data** is therefore one of the most important outcome of semantic Web technologies. The Linked Open Data (LOD) is a set of RDF knowledge graphs published and freely accessible on the Web; they are of various natures relating to many domains such as social networks, science, media, and publications. The LOD-Cloud catalog[1] reports that the growth in the number of integrated datasets has been steadily increasing with very significant increases between 2014 and 2017. Ontologies are the keystone of the semantic Web as they not only enable this linkage of knowledge graphs over the Web but also **logical inference** over knowledge graphs, i.e., the automatic deduction of implicit statements in a graph based on the axioms of ontologies.

An ontology has several definitions from both a philosophical and a computational perspective [2]: *"An ontology is a formal and explicit specification of a shared conceptualization"* is a concise definition highlighting its purpose. An ontology is composed of *a set of classes* and *a set of relations* representing the concepts of the domain knowledge, *a set of individuals that are instances of classes*, and *a set of axioms* involving classes, relations and individuals and capturing domain knowledge. OWL is the semantic Web language to represent ontologies for RDF graphs.

The quality of ontologies and their enrichment are major research issues: the development of the semantic Web requires to extract ontological knowledge from the available information sources and to evaluate the quality of the constructed ontologies before using them to infer new facts. Ontology validation, quality control and enrichment are the subject of many research works [3]–[7]. Ontology learning has an important place in the development of the semantic Web as it allows the enrichment of ontologies from assertions in RDF knowledge graphs of the LOD. Despite the massive development of knowledge graphs in the LOD there is a lack of rich ontologies and the quality of the available knowledge on the LOD is still an issue. For instance, although *DBpedia* is one of the reference knowledge graph of the LOD, constructed with information extracted from Wikipedia, its ontology is sketchy and poor in axioms. Our work fits in the field of

[1]https://lod-cloud.net/

ontology learning. We started from an evolutionary approach previously proposed to extract OWL axioms from an RDF knowledge graph [8]–[10]. In this approach, the extraction of candidate axioms relies on an evolutionary algorithm and their evaluation on SPARQL queries to compute a score based on possibility theory [11]. Focusing on the evaluation of subsumption axioms of the form $\texttt{SubClassOf}(C, D)$, where C and D are class expressions, a possibility theory-based scoring involves a significant computational cost, in terms of resources and time. This limits the practical applicability of this approach. In this paper, we present three novel contributions to overcome these limitations:

A. **a multi-threading system** to parallelize OWL axioms assessment, especially the computation of exceptions to axioms,
B. **an extension of the original heuristic to avoid redundant computation**, with an explanation of the computational problem,
C. **an optimization of SPARQL query chunking** relying on an extension of SPARQL 1.1 Federated Query to automatically iterate a SPARQL federated query service call.

In order to assess these contributions, we tested the integrity of the results and analyzed their impact in term of CPU time saving, using the previous work [12] as a benchmark. The source code of the project is available on GitHub.[2]

This paper is organized as follows: In Section II we summarize the principles of the approach to extract OWL axioms from RDF data to the improvement of which we contribute. We present our contributions in Section III and the results of the experiments that we conducted to evaluate them in Section IV. We conclude in Section V.

II. Preliminaries

A. OWL SubClassOf Axioms

Six categories of OWL axioms can be distinguished: **Class expression** axioms (SubClassOf, DisjointClasses, …), **Object property expression** axioms (SubObjectPropertyOf, DisjointObjectProperties, …), **Data property expression** axioms (SubDataPropertyOf, DisjointDataProperties, …), **Datatype definition** axioms (DatatypeDefinition), **Keys** axioms (HasKey) and **Assertion** axioms (ClassAssertion, ObjectPropertyAssertion). In this paper, we focus on the **SubClassOf** class expression axiom [13]. Its OWL functional-style syntax is `SubClassOf(`C D`)`, where C and D are OWL classes occurring in an RDF graph, e.g. `dbo:Organisation`, `dbo:Work`, `dbo:Plant` in the DBpedia knowledge graph with `dbo` the prefix denoting the DBpedia ontology namespace.

Using the $\mathcal{SHOIQ}$ description logic syntax, the notation $C \sqsubseteq D$ highlights the inclusion of C into D, i.e. the fact that instances of C are also instances of D. This is described in the direct model-theoretic semantics of OWL through the notation $C^I \subseteq D^I$ where I represents individuals in a knowledge graph. SubClassOf axioms are used to make up the taxonomic backbone of most ontologies, such as the *DBpedia 2015-04 Ontology*. Here are some examples:

- `dbo:Actor` $\sqsubseteq$ `dbo:Artist`,
- `dbo:Agglomeration` $\sqsubseteq$ `dbo:PopulatedPlace`,
- `dbo:Annotation` $\sqsubseteq$ `dbo:WrittenWork`

B. An Evolutionary Approach to OWL Axiom Extraction

Our research area focuses on **Axiom Learning** [9], which is a bottom-up approach, using learning algorithms and relying on instances from several existing knowledge and information resources to discover axioms. Axiom learning algorithms can help reduce the overall cost of axiom extraction and ontology construction in general.

To this aim, we use an evolutionary approach, namely **Grammatical Evolution** [8]. Using a predefined grammar in BNF format, we can generate a random set of candidate axioms, formed according to the syntax defined in the **BNF** file. Of course, this simple process is not sufficient to obtain an axiom that is meaningful. For this purpose, we use an evolutionary process based on this grammar to allow the generation of random candidate axioms, which together form a population, and the evaluation of this population using a *fitness* function, which we aim to maximise. This process, which iterates according to the given parameters, allows us to obtain from initially random candidate axioms, new candidate axioms that are more and more consistent, i.e., which present a non-zero fitness, and these individuals will be taken as models by the algorithm to generate a new population of axioms which inherit traits from the best individuals.

It is of paramount importance, for such approach to work correctly, that the fitness estimated for a candidate axiom accurately captures its compatibility with the facts asserted in the knowledge graph at hand. To this aim, we adopt am axiom evaluation heuristic, based on possibility theory [11], which has been shown to be particularly reliable in view of the open-world semantics of RDF knowledge graphs [13]. The heuristic computes a **possibility** and a **necessity** for a given axiom; some particular cases, like the `DisjointClasses` axiom, require a slightly different treatment, whereby their necessity is always zero and only their possibility is computed [8], [9].

C. A Possibilistic Heuristic to Evaluate SubClassOf Axioms

Taking inspiration from possibility theory [11], we previously proposed a possibilistic heuristic to assess `SubClassOf` axioms with the two possibilistic metrics of **possibility** and **necessity** of an axiom [10], [12]–[14]. Possibility theory is a mathematical theory of epistemic uncertainty which uses the events, variables, … denoted ω of a universe of discourse Ω ($\omega \in \Omega$) where each ω has a degree of possibility such that $\pi : \Omega \to [0, 1]$.

In order to assess the possibility and necessity of an axiom ϕ, the heuristic considers υ_ϕ^+, the number of confirmations observed among the elements of υ_ϕ, the support of ϕ, and υ_ϕ^-,

[2]https://github.com/RemiFELIN/RDFMining

the number of exceptions observed. It defines the possibility $\Pi(\phi)$ and necessity $N(\phi)$ of the axiom as follows:

$$\Pi(\phi) = 1 - \sqrt{1 - \left(\frac{\upsilon_\phi - \upsilon_\phi^-}{\upsilon_\phi}\right)^2},$$

$$N(\phi) = \begin{cases} \sqrt{1 - \left(\frac{\upsilon_\phi - \upsilon_\phi^+}{\upsilon_\phi}\right)^2}, & \text{if } \Pi(\phi) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

To decide the acceptance of an axiom according to its possibility and necessity, an Acceptance/Rejection Index (ARI) is defined, combining the two measures:

$$ARI(\phi) = N(\phi) + \Pi(\phi) - 1 \in [-1, 1].$$

Thus, an ARI whose value is less than 1 indicates that there is at least 1 exception υ_ϕ^- for an axiom ϕ. When the ARI is equal to 0, we are in a case of **total ignorance** for ϕ: this indicates that $\upsilon_\phi^- \to \emptyset$ and $\upsilon_\phi^+ \to \emptyset$. An ARI value greater than 0 implies $\upsilon_\phi^- \to \emptyset$ and confirmations founded. Finally, a perfect axiom has an ARI equal to 1 and implies $\upsilon_\phi^- \to \emptyset$ and $\upsilon_\phi^+ \to \upsilon_\phi$.

The implementation of the above formulas was carried out in SPARQL. The queries presented in Figures 2 and 3. return (respectively) the number of confirmations υ_ϕ^+ and the number of exceptions υ_ϕ^- for a given subsumption axiom $C \sqsubseteq D$. The computation of the number of confirmations is quite simple: we count the number of instances belonging to both the subclass C and the superclass D.

On the other hand, the calculation of the number of exceptions is trickier, since, according the the open-world hypothesis, if a fact is not asserted in a knowledge base, this does not necessarily mean it is false. Therefore, an exception is assumed only when an instance of C is found to also belong in another class that does not share any instance with D. This query, which gives of course still an approximation, even though a much finer one, of the actual number of true exceptions, turns out to be computationally quite expensive.

```
SELECT (COUNT(DISTINCT ?t) AS ?nic) WHERE {
    ?x a <C>, ?t .
}
```

Fig. 1: SPARQL query used to compute the number of intersecting classes (`nic`) for a subclass C.

III. Contributions

A. Multi-Threading System

We implemented a multi-threading system in order to parallelize the evaluation of the axioms, which allows us to significantly reduce the overall computation time. This gain is the more significant the greater is the number of CPU cores available, since the program creates a number of threads equivalent to the number of cores available on the machine on which the software is run. Nevertheless, while this optimization can reduce the latency of axiom evaluation if a large number of cores is available, it does not reduce the overall cost of the task.

```
SELECT (COUNT(DISTINCT ?x) AS ?n) WHERE {
    ?x a <C>, <D> .
}
```

Fig. 2: Naive implementation of our possibilistic heuristic in SPARQL: retrieval of the confirmations for a given axiom `SubClassOf(<C> <D>)`.

```
SELECT (COUNT(DISTINCT ?x) AS ?n) WHERE {
    ?x a <C>, ?t .
    FILTER NOT EXISTS { ?y a ?t, <D> . }
}
```

Fig. 3: Naive implementation of our possibilistic heuristic in SPARQL: retrieval of the exceptions for a given axiom `SubClassOf(<C> <D>)`.

B. A Heuristic to Avoid Redundant Computation

We propose an optimization to reduce the overall computation time of the ARI of an axiom by reducing the computation time of its exceptions. Considering the naive implementation of the retrieval of the exceptions for a given axiom `SubClassOf(<C> <D>)` in the SPARQL query depicted in Figure 3, we can observe the possible and useless repetition of the retrieval of the same types `?t` for different instances `?x` of a subclass `<C>` satisfying a filter condition that does not depend on `?x`. And it is very likely that the same types are found many times for different individuals, implying the repetition of these same computations. As a result, we have split the SPARQL query to compute exceptions to an axiom in two (the latter query being dependent on the result of first one):

1) The first query retrieves, once for all, distinct types (i.e., classes) being evaluated as potentially containing exceptions to the axiom (Figure 4).
2) The second query retrieves the instances that belong to both subclass `<C>` and at least one of the classes retrieved bu the previous query, which suggests them to be considered as exceptions (Figure 5).

Additionally, considering the worst case where (almost) all the instances of subclass `<C>` are exceptions to axiom `SubClassOf <C> <D>` (in that case the axiom's ARI will be close to -1), the computation time of exceptions may be very long if the number of instances of `<C>` is high. The computational cost of `FILTER NOT EXISTS` grows more than linearly with the number of instances that have to be filtered. Therefore, we have also developed a chunking technique for SPARQL queries in order to split the task into

```
SELECT DISTINCT ?t WHERE {
    {
        # We retrieve the other classes of the instances of subclass <C>.
        SELECT ?t WHERE {
            SELECT DISTINCT ?t WHERE { ?x a <C> , ?t . } ORDER BY ?t
        } LIMIT $limit OFFSET $offset
    }
    # From these classes, we remove those sharing instances with superclass <D>.
    FILTER NOT EXISTS { ?z a ?t, <D> . }
}
```

Fig. 4: Implementation of our optimized heuristic in SPARQL: retrieval of the classes which instances are possible exceptions to axiom `SubClassOf(<C> <D>)`.

```
SELECT DISTINCT ?x WHERE {
    ?x a <C>, ?t
    VALUES ?t { <t1> <t2> ... <tn> }
} LIMIT $limit OFFSET $offset
```

Fig. 5: Implementation of our optimized heuristic in SPARQL: retrieval of the exceptions of `SubClassOf(<C> <D>)` as being the instances of the classes computed in Fig. 4.

several steps. The `LIMIT` and `OFFSET` modifiers allow the results of a SPARQL query to be paginated, making it quicker to manipulate a subset. The resulting algorithm is presented in Algorithm 2.

C. Optimizing the Chunking of SPARQL Queries

In general, it is quite tedious to implement chunking of SPARQL queries. Moreover, one may still want to resort to chunking a SPARQL query with a `VALUES` clause, due to the fact that some servers limit the number of elements handled in such a clause. Within the framework of our special use case of computing the exceptions to a `SubClassOf` axiom, we propose a generic SPARQL operator allowing us to automatically integrate the pagination of the results with an iteration system. We build upon previous work on using URL parameters in SPARQL federated query services [15]. This operator allows specifying the iteration with a URL parameter **loop** set to true and the pagination with a URL parameter **limit** set to the chosen number of first results to be returned by a query. The advantage of this operator is twofold: on the one hand, it makes it easier to code the iteration and chunking of SPARQL queries; on the other hand, the iteration and chunking are delegated to the SPARQL engine, which in general is more effective at performing them than the user code calling the engine several times. The algorithm of this operator is described in Algorithm 1. It is implemented into the *Corese*[3] semantic Web factory [16].

By using this novel `loop+page` operator, we propose another algorithm to compute the exceptions to a `SubClassOf`

[3]https://github.com/Wimmics/corese

Algorithm 1 Iterate and page a SPARQL query

```
1:  sol ← {}
2:  q ← the body of a SPARQL query
3:  for i=$start; i<=$until; i++ do
4:      q ← q+LOOP=$true&LIMIT=$limit&OFFSET=i*$limit
5:      res ← eval(SERVICE url{q})
6:      if len(res) == 0 then
7:          break
8:      end if
9:      sol ← sol ∪ res
10: end for
11: return sol
```

`<C> <D>`axiom. Both the query to retrieve the classes potentially containing exceptions (Figure 6) and the query to retrieve the exceptions (Figure 7) are SPARQL federated queries using the parameters `loop` and `limit` in the URL of the remote query service in the `SERVICE` clause. The resulting algorithm is detailed in Algorithm 3). When compared to Algorithm 2, it is obviously less tedious to set up.

IV. Experiments

In order to evaluate our contributions, we carried out the scoring of 722 candidate axioms against an RDF dataset extracted from *DBpedia 3.9* comprising 463,343,966 triples and 532 OWL classes. This experiment has two objectives: on the one hand, to show that our optimizations give results that are faithful to the initial results achieved with the original heuristic presented in [12], and, on the other hand, to highlight the computation time savings obtained. The experiments were performed on a server equipped with an Intel(R) Xeon(R) CPU E5-2637 v2 processor at 3.50GHz clock speed, with 172 GB of RAM, 1 TB of disk space running under the Ubuntu 18.04.2 LTS 64-bit operating system.

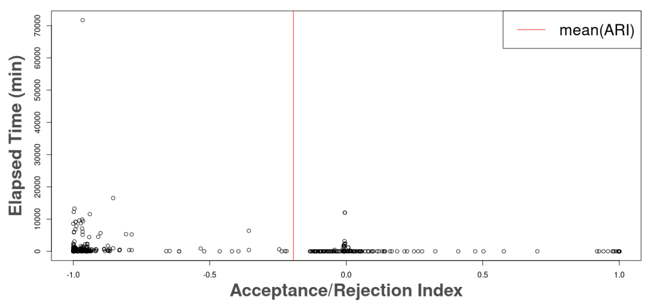
As it can be seen in Fig. 8, when using our optimization, the computation time is significantly reduced, with a maximum computation time reduced from 71,699 to 489 minutes. The ARIs values computed for each axiom remain unchanged, giving the same average ARI value ($\sim$ -0.1936). As already reported in the experiments described in [14], with an ARI

```
SELECT DISTINCT ?t WHERE {
    SERVICE <$url/sparql?loop=true&limit=$limit> {
        SELECT DISTINCT ?t WHERE { ?x a <C>, ?t . }
    }
    SERVICE <$url/sparql> {
        VALUES ?t {undef}
        FILTER NOT EXISTS { ?z a <D>, ?t . }
    }
}
```

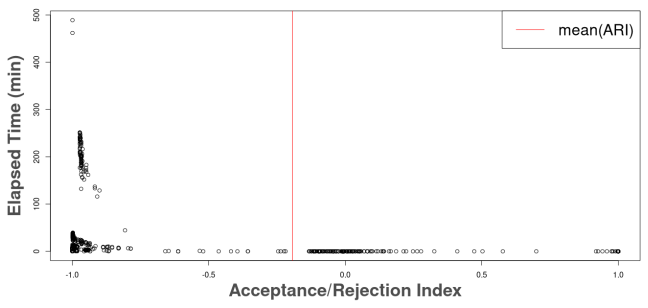
Fig. 6: Second implementation of our optimized heuristic with a SPARQL federated query using parameters `loop` and `limit`: retrieval of the classes which instances are possible exceptions to axiom `SubClassOf(<C> <D>)`.

```
SELECT DISTINCT ?x WHERE {
    SERVICE <$url/sparql?loop=true&limit=$limit> {
        ?x a <C>, ?t VALUES ?t { <t1> <t2> ... <tn> }
    }
}
```

Fig. 7: Second implementation of our optimized heuristic with a SPARQL federated query using parameters `loop` and `limit`: retrieval of the exceptions of `SubClassOf(<C> <D>)` as being the instances of the classes computed in Fig. 6.



(a) Original heuristic



(b) Results obtained with contributions A+B

Fig. 8: Comparison of the ARI values of 722 axioms computed against DBpedia 3.9 using the original heuristic in [12] and our proposed optimization

value greater than 1/3 as an acceptance criterion for an axiom ϕ, the number of accepted axioms is only 197 against 525 rejected (27,28% of acceptance rate). Here are interesting example axioms with ARI = 1 that are not already in the *DBpedia 3.9* ontology and could be considered to enrich it: `SubClassOf(dbo:Chef dbo:Agent)`, `SubClassOf(dbo:Venue dbo:Place)`, `SubClassOf(dbo:RadioHost dbo:Person)`.

We also compared the computation times of the ARI of each of the 722 candidate axioms when using the original heuristic or our proposed optimization. Figure 9 presents the initial computation time of the axioms ARIs when using the original heuristic as a function of the computation time when using our proposed optimization. This highlights the huge proportion of axioms for which our proposed optimization saves computation time and the importance of this time saving.

The average CPU computation time for evaluating an axiom is 30 minutes with our proposed optimization compared to 578 minutes with the original heuristic, with a significant average time saving of 548 minutes. For most axioms we observe a lower computation time: 593 axioms are faster to evaluate with our method, i.e. 82% of the candidate axioms tested. In particular, our optimization solves the problem of extremely long CPU computation times for some axioms (see Figure see Figure 8a, up to 71,699 minutes). Our optimisation significantly reduced the computational cost to a maximum value of 489 minutes, a reduction by a factor of approximately 150. For 129 axioms, i.e. 18% of the tested candidate axioms, we observe a higher computation time with our optimization. This occurs for axioms where the instances of `<C>` do not share any of their other types. In that case, the execution of the initial single SPARQL query (Figure 3) is faster than the

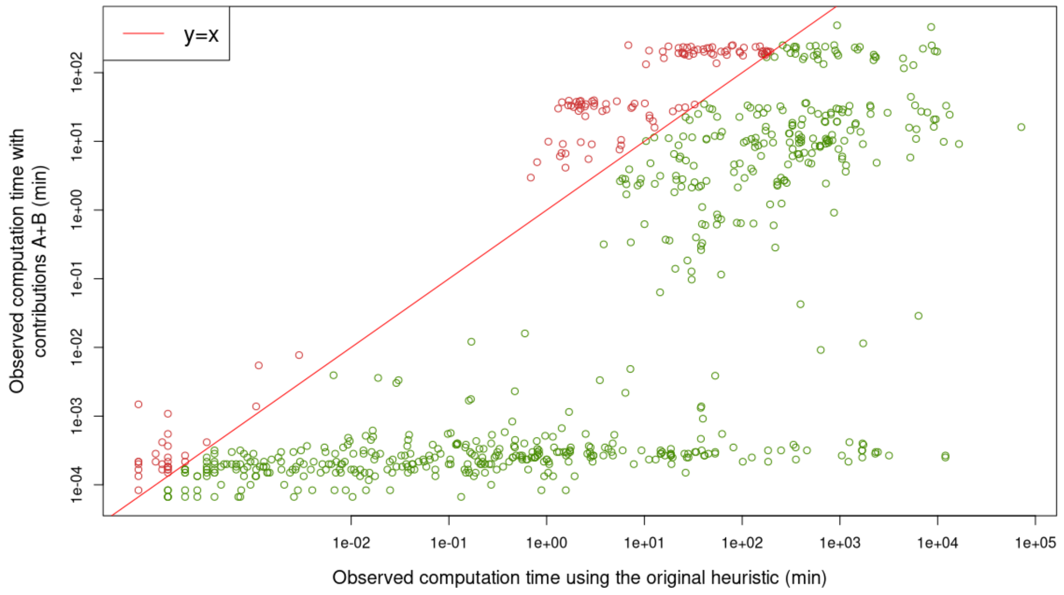


Fig. 9: Comparison of the computation times (CPU) of axioms ARIs with the original heuristic and with our proposed optimization (A+B), highlighting the proportion of axioms for which our optimization saves time (in green) or loose time (in red). Both axes are logarithmic.

execution of the two SPARQL queries in our optimisation (Figures 6 and 7). However, the results show an average increase in computation time of about 57 minutes and a maximum increase of 244 minutes for these axioms. These are much smaller losses compared to the average and maximum gain we get on the other 82% candidate axioms.

To evaluate our last optimization using a loop+page operator relying on the extension of SPARQL federated query with URL parameters, we compared the times to compute axioms ARIs with this extension and with standard SPARQL queries. The results presented in Fig. 10 highlight a significant CPU time saved when using SPARQL federated queries with URL parameters. The average computation time for an axiom is reduced by approximately 12 minutes. There are 683 axioms quicker to assess using the loop operator and only 39 axioms for which there is a minor increase of the computation time. This observed loss of time is explained by a significant accumulation of HTTP requests submitted to the server in the case where the potential exceptions have a high number of types. This accumulation increases, even more, when several axioms are evaluated simultaneously (contrib. A). A large number of HTTP requests can increase server latency, however, we can see that these time losses are marginal. This suggests that the implementation of this operator optimises the chunking of queries internally, making the computation cost less important than our first proposal, in which we truncate the queries (see Algorithm 2).

V. Conclusion

We presented an optimisation of a possibilistic heuristic approach for the assessment of OWL SubClassOf axioms against RDF data, enabling a significant CPU computation time saving and offering new perspectives to this approach. Our contribution lies in the proposition and the combination of a multi-threading system and two algorithms to compute the exceptions to candidate SubClassOf axioms: a first one relying on a classical approach for SPARQL query chunking and a second one where we put forward our expertise with the *Corese* semantic Web factory to propose a loop+page operator implemented with URL parameters for SPARQL federated queries.

As future work, we plan to adapt the proposed optimizations to the evaluation of other types of OWL axioms. Another challenge is to generalize this optimisation to computational problems with SPARQL queries having a similar form, and thus bring the benefit of our work to the semantic Web community.

Acknowledgments

This work has been partially supported by the French government, through the 3IA Côte d'Azur "Investments in the Future" project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

[1] P. Pieter, Z. Sijie, and L. Yong-Cheol, "Semantic web technologies in aec industry: A literature overview," *Automation in Construction*, vol. 73, pp. 145–165, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580516302928

[2] N. Guarino, D. Oberle, and S. Staab, *What Is an Ontology?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–17. [Online]. Available: https://doi.org/10.1007/978-3-540-92673-3_0

[3] D. Vrandečić, *Ontology Evaluation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 293–313. [Online]. Available: https://doi.org/10.1007/978-3-540-92673-3_13

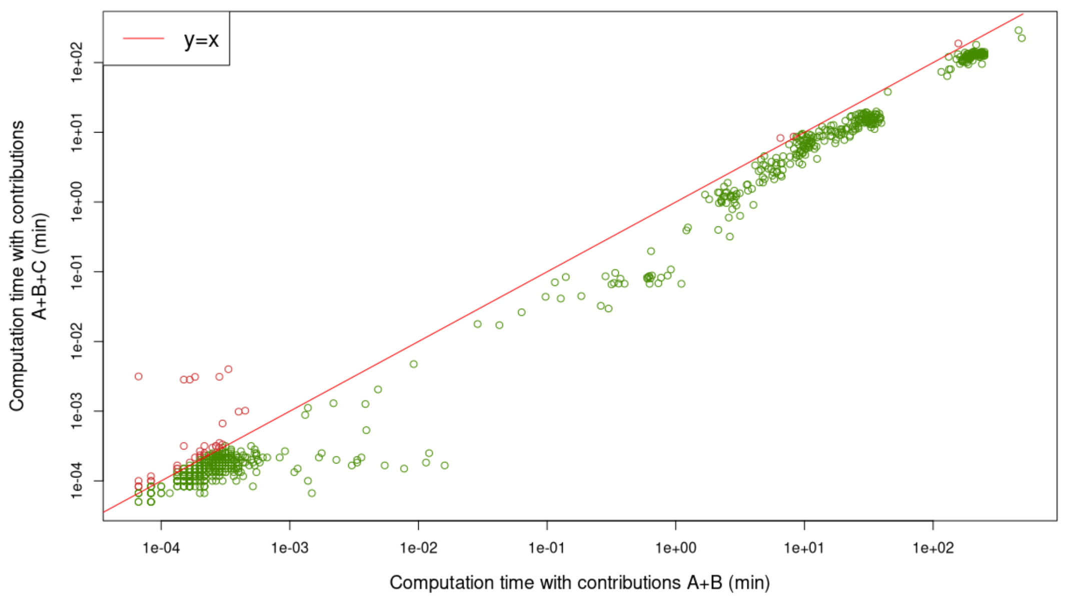


Fig. 10: Comparison of the computation times (CPU) of axioms ARIs with our first (A+B) and second (A+B+C) proposed optimizations, highlighting the proportion of axioms for which for which our last optimization saves (in green) or loose (in red) time with A+B+C. Both axes are logarithmic.

[4] J. Raad and C. Cruz, "A survey on ontology evaluation methods," in *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, ser. IC3K 2015. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2015, p. 179–186. [Online]. Available: https://doi.org/10.5220/0005591001790186

[5] S. Mc Gurk, C. Abela, and J. Debattista, "Towards ontology quality assessment." in *MEPDaW/LDQ@ ESWC*, 2017, pp. 94–106.

[6] G. Petasis, V. Karkaletsis, G. Paliouras, A. Krithara, and E. Zavitsanos, *Ontology Population and Enrichment: State of the Art*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 134–166. [Online]. Available: https://doi.org/10.1007/978-3-642-20795-2_6

[7] L. Bühmann and J. Lehmann, "Universal owl axiom enrichment for large knowledge bases," in *Knowledge Engineering and Knowledge Management*, A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles, and N. Hernandez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 57–71.

[8] T. H. Nguyen and A. G. Tettamanzi, "An evolutionary approach to class disjointness axiom discovery," in *IEEE/WIC/ACM International Conference on Web Intelligence*, ser. WI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 68–75. [Online]. Available: https://doi.org/10.1145/3350546.3352502

[9] T. H. Nguyen and A. G. B. Tettamanzi, "Using grammar-based genetic programming for mining disjointness axioms involving complex class expressions," in *Ontologies and Concepts in Mind and Machine*, M. Alam, T. Braun, and B. Yun, Eds. Cham: Springer International Publishing, 2020, pp. 18–32.

[10] R. Felin and A. G. Tettamanzi, "Using grammar-based genetic programming for mining subsumption axioms involving complex class expressions," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, ser. WI-IAT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 234–240. [Online]. Available: https://doi.org/10.1145/3486622.3494025

[11] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 100, pp. 9–34, 1999.

[12] A. G. Tettamanzi, C. Faron-Zucker, and F. L. Gandon, "Dynamically time-capped possibilistic testing of subclassof axioms against rdf data to enrich schemas," in *K-CAP*, ser. Proceedings of the 8th International Conference on Knowledge Capture, K. Barker and J. M. Gómez-Pérez, Eds., no. 7, Palisades, NY, United States, October 2015.

[13] A. Tettamanzi, C. Faron Zucker, and F. Gandon, "Possibilistic testing of OWL axioms against RDF data," *International Journal of Approximate Reasoning*, 2017. [Online]. Available: https://hal.inria.fr/hal-01591001

[14] A. G. Tettamanzi, C. Faron-Zucker, and F. Gandon, "Testing owl axioms against rdf facts: A possibilistic approach," in *Knowledge Engineering and Knowledge Management*, K. Janowicz, S. Schlobach, P. Lambrix, and E. Hyvönen, Eds. Cham: Springer International Publishing, 2014, pp. 519–530.

[15] O. Corby, C. Faron, F. Gandon, D. Graux, and F. Michel, "Beyond Classical SERVICE Clause in Federated SPARQL Queries: Leveraging the Full Potential of URI Parameters," in *WEBIST 2021 - 17th International Conference on Web Information Systems and Technologies*, Online, Portugal, Oct. 2021. [Online]. Available: https://hal.inria.fr/hal-03404125

[16] O. Corby and C. F. Zucker, "Corese: A corporate semantic web engine," in *International Workshop on Real World RDF and Semantic Web Applications, International World Wide Web Conference*, 2002.

Algorithm 2 Compute exceptions to a `SubClassOf` axiom according to contribution B.

Output: *numExceptions, and a list of these exceptions.*
Require: $numConfirmations \neq referenceCardinality$

```
 1: $q_1 \leftarrow$ SPARQL query presented in Fig. 1
 2: $q_2 \leftarrow$ SPARQL query presented in Fig. 4
 3: $offset \leftarrow 0$
 4: $limit \leftarrow 1000$
 5: $exceptions \leftarrow \{\}$
 6: $types \leftarrow \{\}$
 7: $nic \leftarrow eval(q_1)$
 8: while $offset \neq nic$ do
 9:     $q_2 \leftarrow q_2 + LIMIT\ limit\ OFFSET\ offset$
10:     $types \leftarrow types \cup eval(q_2)$
11:     $offset \leftarrow offset + min(nic - offset, limit)$
12: end while
13: $start \leftarrow 0$
14: $step \leftarrow 100$
15: $limit \leftarrow 10000$
16: while $start \neq \|types\|$ do
17:     $offset \leftarrow 0$
18:     $end \leftarrow start + min(step, \|types\|)$
19:     while true do
20:         $q_3 \leftarrow$ SPARQL query presented in Fig. 5
21:         using VALUES { $t_i \in types,\ i \in [start, end]$ }
22:         $q_3 \leftarrow q_3 + LIMIT\ limit\ OFFSET\ offset$
23:         $e \leftarrow eval(q_3)$
24:         $exceptions \leftarrow exceptions \cup e$
25:         if $\|e\| = limit$ then
26:             $offset \leftarrow offset + limit$
27:         else break
28:         end if
29:     end while
30:     $start \leftarrow start + min(\|types\| - start, step)$
31: end while
32: $numExceptions \leftarrow \|exceptions\|$
```

Algorithm 3 Compute exceptions to a `SubClassOf` axiom using contributions B and C.

Output: *numExceptions, and a list of these exceptions.*
Require: $numConfirmations \neq referenceCardinality$

```
 1: $limit \leftarrow 1000$
 2: $q_1 \leftarrow$ SPARQL query presented in Fig. 6
 3: $types \leftarrow eval(q_1)$
 4: $start \leftarrow 0$
 5: $step \leftarrow 50$
 6: $limit \leftarrow 10000$
 7: while $start \neq \|types\|$ do
 8:     $end \leftarrow start + min(step, \|types\|)$
 9:     $q_2 \leftarrow$ SPARQL query presented in Fig. 7
10:     using VALUES { $t_i \in types,\ i \in [start, end]$ }
11:     $exceptions \leftarrow exceptions \cup eval(q_2)$
12:     $start \leftarrow start + min(\|types\| - start, step)$
13: end while
14: $numExceptions \leftarrow \|exceptions\|$
```