Optimizing the Computation of a Possibilistic Heuristic to Test OWL 2 SubClassOf Axioms Against RDF Data

***Abstract*—The growth of the Semantic Web requires various tools to manage data, make them available to all and use them for a wide range of applications. Among these research areas, Ontology enrichment took an important place where many contributions provide solutions using AI, reasoners, etc... The discovery of OWL 2 Axioms is one of the tasks of the Ontology Enrichment domain. To this end, we proposed a possibilistic framework to enrich ontologies with OWL 2 Axioms extraction using an evolutionary approach. In the context of this approach, the assessment of this type of axioms involves a significant computational cost, especially in terms of computation time. In this paper, we develop an optimisation of this possibilistic heuristic approach for the OWL 2 SubClassOf axioms assessment, particularly for exception extraction. We verify that the proposed approach is equivalent to the previous one and carry on a comparative analysis. We also propose a solution deployed on a semantic web tool to apply our approach and we will analyse it. We will conclude with the proposed approach and the new perspectives of this work.**

***Index Terms*—Knowledge Graphs, OWL 2 Axioms, Ontology Enrichment**

I. Introduction

Over time, the semantic Web has developed significantly through the development of W3C standardised technologies such as RDF (Resource Description Framework), OWL (Web Ontology Language), SHACL (Shapes Constraint Language), ...As a result, the use cases have diversified widely and satisfy 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 these data through the web. These data are linked together by various relationships that make sense of the data set, which we call a knowledge graph. The second axis: **linked data**, is therefore one of the most important advantages of the semantic web. the best known example is the LOD (Linked Open Data) which is a set of datasets of various natures relating to many domains such as social networks, science, media, publications,...The official website[1] reports that the growth in the number of integrated datasets has been steadily increasing with very significant increases between 2014 and 2017. Finally, **logical inference** using ontologies allows the automatic induction of new knowledge. This is a considerable advantage if the ontologies in question are correct and complete.

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. In this context, an Ontology is composed of four parts : *a set of concepts* which describes in an abstract way the elements that are part of the knowledge domain, *a set of relations* between elements, *a set of instances* and *a set of axioms*. The RDF standard allows such data to be written in XML format (other languages exist such as Turtle) and OWL provides a representation language for building semantic web ontologies.

The quality of ontologies and the enrichment of knowledge are major research issues: we need to be able to translate information already present in terms of meaning and to be able to evaluate the quality of this information before deducing new information. Some domains such as validation, quality control of ontologies and Ontology Enrichment are the subject of many contributions [3]–[7]. Likewise, Ontology Learning has an important place in the development of the LOD as it allows the enrichment of ontologies through a set of research domains. Continuous learning is particularly important when using dynamic data sources where a large amount of data is subject to real-time changes. Therefore, continuous learning and inference of new knowledge, through consistency evaluation, seems to be a major issue in ontology enrichment. Despite the massive development of knowledge and LOD with the purpose of information extraction and formatting systems, there is a certain lack of information in ontologies and the quality of information present in the semantic Web is not guaranteed. Indeed, if we take one of the reference bases of the LOD, namely *DBPedia*, which is a knowledge base constructed based on information extracted from Wikipedia, it presents some shortcomings in terms of information, especially in the axiom sets of the ontology. Our work fits in the second domain, where we propose a software able to extract from a knowledge graph and an ontology, axioms whose principles and their representations are provided by the OWL 2 representation language.

In our research, the extraction and evaluation of candidate axioms is done through the use of an evolutionary algorithm

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

(to be discussed later) and the use of SPARQL queries (SPARQL Protocol and RDF Query Language) in order to apply our evaluation method based on possibility theory [8]. Focusing on the evaluation of candidate subsumption axioms of the form `SubClassOf`(C, D), where C and D are class expressions, possibility theory-based evaluation involves a significant computational cost, in terms of resources and time. This limits the practical applicability of our solution.

In this paper, we present the background of our research on the evaluation of these axioms, then we provide a contribution based on an implementation optimization of our heuristic, which answers the problem in a satisfactory way, bringing about a dramatic reduction of its computational cost. We then analyse the impact of this heuristic, especially in terms of CPU time. Finally, we study this heuristic in the extraction domain and more precisely the extraction of subsumption axioms involving complex class expressions.

II. Preliminaries

In order to simplify reading, we propose to introduce the context of our research area and useful bases for understanding.

A. OWL 2 SubClassOf Axioms

The *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*[2] provides a complete definition of this language. We distinguish 6 categories of OWL 2 Axioms : **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 subsumption axioms **SubClassOf** [9] provided by the OWL 2 standard. Its OWL 2 functional syntax is `SubClassOf(`C D`)` where C and D represent respectively concepts of the studied knowledge graph, let's take the example of DBPedia, the concepts can be the following:
Consider the following prefix `dbo` → `<http://dbpedia.org/ontology/>`.
`dbo:Organisation`, `dbo:Work`, `dbo:Plant`, ...

Using the $\mathcal{SHOIQ}$ description logic syntax, the notation $C \sqsubseteq D$ highlight the perfect affiliation of the instances of C with the instances of D, and so the instances of C are also instances of D, this is described semantically by the notation $C^I \subseteq D^I$ where I represents individuals in a knowledge graph. This type of axioms are present in the ontologies such as the *DBPedia 2015-04 Ontology*, Here are some examples:
`SubClassOf(dbo:Actor dbo:Artist)`,
`SubClassOf(dbo:Agglomeration dbo:PopulatedPlace)`,
`SubClassOf(dbo:Annotation dbo:WrittenWork)`,
...

B. RDFMiner project

Our research area focuses on **Axiom Learning** [10], 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** [11]. Using a predefined grammar in BNF format, we can generate candidate axioms, formed according to the syntax defined in the **BNF** file, at random. 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 model by the algorithm to generate a new population of axioms which inherit traits from the best individuals.

In order to be able to evaluate candidate axioms satisfactorily, we base our heuristic on possibility theory [8] to calculate the **possibility** and **necessity** for a given axiom, some particular case especially for disjunction axioms `DisjointClasses` must be treated differently [10], [11]. Indeed, this theory allows us to overcome the lack of information in the datasets and the unguaranteed quality which, with a probabilistic heuristic, has a negative impact on the results.

We have chosen to work with OWL 2 axioms in order to satisfy two objectives:

- Extracting new knowledge from an existing knowledge base expressed in RDF.
- Injecting such extracted knowledge into an ontology in order to be able to use it to infer its logical consequences.

The format is appropriate for these two distinct purposes as it is very expressive and is suitable for injection into an ontology [9]. The source code of the project is available on GitHub[3]

[2]https://www.w3.org/TR/owl2-syntax/

[3]https://anonymous.4open.science/r/RDFMining-9CBC/README.md

C. A possibilistic heuristic to evaluate SubClassOf axioms

Based on the work of *Lotfi A. Zadeh* [8], various works [9], [12]–[14] have proposed a possibilistic heuristic to assess `SubClassOf` axioms with possibilistic metrics such as the notion of **possibility** and the **necessity** of an axiom. 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 $\pi : \Omega \rightarrow [0, 1]$.

In order to assess the possibility and necessity of an axiom ϕ, we consider υ_ϕ^+ the confirmations observed among the elements of υ_ϕ and υ_ϕ^- the conterexamples observed. We define $\Pi(\phi)$ and necessity $N(\phi)$ 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, we propose a third indicator : **ARI** (Acceptance/Rejection Index) such as:

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

Thus, an ARI whose value is less than 1 indicates that there are at least 1 exceptions 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 $\upsilon_\phi^- \rightarrow \emptyset$ and $\upsilon_\phi^+ \rightarrow \emptyset$. Finally, an ARI value greater than 0 implies $\upsilon_\phi^- \rightarrow \emptyset$ and confirmations founded. a perfect axiom has an ARI equal to 1 and implies $\upsilon_\phi^- \rightarrow \emptyset$ and $\upsilon_\phi^+ \rightarrow \upsilon_\phi$.

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

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

The implementation of the formulas seen previously was carried out using SPARQL queries and are presented in Figures 2 and 3 and returns (respectively) the number of confirmations υ_ϕ^+ and the number of exceptions υ_ϕ^-. The complexity of the calculation of the confirmations does not cause any particular computational difficulties since we are checking for instances belonging to both the subclass C and the superclass D.

```
SELECT (count(distinct ?x) AS ?n) WHERE
{
    ?x a <C> .
    ?x a <D>
}
```

Fig. 2: Implementation of our possibilistic heuristic in SPARQL to compute confirmations for a given axiom `SubClassOf(<C> <D>)`

```
SELECT (count(distinct ?x) AS ?n) WHERE
{
    ?x a <C> , ?t .
    FILTER NOT EXISTS {
        ?z_1 a ?t , <D> .
    }
}
```

Fig. 3: Implementation of our possibilistic heuristic in SPARQL to compute exceptions for a given axiom `SubClassOf(<C> <D>)`

We note that the calculation of the types (`?t`) is done for each individual, which represents an important move and subsequently involves the calculation of the individuals that are linked to it (`?z_1`). Moreover, it is very possible that the same types are found several times between individuals, which sometimes implies the repetition of these same calculations.

This analysis has allowed us to identify several directions for minimising the computation time of exceptions. a multi-threading system has been implemented in order to parallelize the evaluation of the axioms, which allows us to significantly reduce the overall computation time. This gain is even more important depending on the number of threads available, the program takes the number of threads equivalent to the number of threads available on the machine on which the software is used. Nevertheless, we realize that this optimization is in a global sense satisfactory but does not affect in any way the costly computation time of the `SubClassOf` axioms

III. Contributions

A. An optimized heuristic using SPARQL Queries

The objective of this optimisation is essentially oriented towards reducing the calculation time of exceptions, which obviously involves reducing the number of calculations when possible. However, we have seen that the discovery of several identical types implies a repetition of the calculations, notably to return the individuals of these same types. Thus, we have split the request into two parts (the latter is dependent on the first):

1) Recovering distinct types (i.e., classes) being evaluated as potentially containing exceptions (Figure 4).
2) Retrieve the instances that belong to the subclass and that, at the same time, belong to at least one of the classes just retrieved, which suggests them to be considered as exceptions (Figure 5).

The calculation of exceptions takes longer when the axiom has an ARI value close to -1. This happens because, in that case, almost all the instances of the subclass represent exceptions, which does not pose a real problem when the number of instances is relatively small, but implies a very high computation time when the number of instances reaches considerable values. Therefore, we have also developed truncation techniques for SPARQL queries in order to split the tasks into several steps. The LIMIT and OFFSET keywords allow the results of a SPARQL query to be paginated and thus truncated, making it quicker to manipulate a subset. The full detail of the implementation is presented in Algorithm 1.

Algorithm 1 Test a `SubClassOf` axiom (optimisation)

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

$offset \leftarrow 0$
$limit \leftarrow 1000$
$exceptions \leftarrow \{\}$
$types \leftarrow \{\}$
$nic \leftarrow$ results of SPARQL Request presented in Fig.1
while $offset \neq nic$ **do**
    compute the types t_i thanks to the request in Fig. 4 from $limit$ to $offset$.
    $types$.insert(t_i)
    $offset \leftarrow offset + min(nic - offset, limit)$
end while
$start \leftarrow 0$
$step \leftarrow 100$
$limit \leftarrow 10000$
while $start \neq len(types)$ **do**
    $offset \leftarrow 0$
    $end \leftarrow min(start + k, len(types))$
    while true **do**
        compute the instances e corresponding to request in Fig. 5 using t_n subset for $n \in [start, end]$, from $limit$ to $offset$
        if $e \notin exceptions$ **then**
            $exceptions$.insert(e)
        end if
        if $len(e) == limit$ **then**
            $offset \leftarrow offset + limit$
        else break
        end if
    end while
    $start \leftarrow start + min(len(types) - start, k)$
end while
$numExceptions \leftarrow len(exceptions)$

B. A `loop` operator to automate iterations in SPARQL queries

It can be seen that it is quite tedious to implement truncation of SPARQL queries. Within the framework of this work, we worked on an operator allowing to automatically integrate the pagination of the results with a system of iteration. This operator allows you to specify the maximum limit l of results returned by a query, the loop operator tool to page the first l results and if the number of results is equivalent to the limit, the operator takes care of executing the query by passing to the following results. For example, if for a given SPARQL query, we need to find 2000 results with a limit $l = 500$, the engine will first page the first 500 results, as the limit is reached, a query is executed to page the next 500 results, ... until 2000 results are reached.

The `loop` operator was integrated into the *Corese* project[4] [15]: *Corese* is a software platform implementing and extending the standards of the Semantic Web such as RDF, RDFS, SPARQL 1.1, SHACL, ... It allows to create, manipulate, parse, serialize, query, reason and validate RDF data, and provides an extended version of the SPARQL standard with the LDScript language [16].

We propose an implementation of this solution (see Algorithm 2) and we notice that it is less tedious to set up, we still have recourse to a truncation for the second request due to errors of the server, this one risks to return intermediate results for a certain set of axioms and thus give a false result.

IV. Experiments

In order to validate our optimisation, we have chosen to carry out a new evaluation of the axioms used in [12], where a list of 722 candidate axioms has been evaluated against RDF Data from *DBPedia 3.9* without time-capping. This dataset is composed of 463,343,966 RDF Triples and provides 532 OWL Class.

This experiment has two objectives: to show that our optimization gives results that are faithful to the initial results and finally to highlight the computation time saving 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.

According to Fig. 8, the time is significantly less in the graph on the right, where the scale has been considerably reduced from a maximum value of 71,699 to 489 minutes.

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

```
SELECT distinct(?t) WHERE {
    {
        # We retrieve the other types related to
        # individuals belonging to the subclass <C>
        SELECT ?t WHERE {
            {
                SELECT distinct(?t) WHERE {
                    ?x a <C> , ?t
                } ORDER BY ?t
            }
        } LIMIT $limit OFFSET $offset
    }
    # For these types, we look for types for which there
    # are no individuals that belong to them and to the
    # superclass <D>.
    FILTER NOT EXISTS {
        ?z a ?t , <D>
    }
}
```

Fig. 4: Implementation of our optimized heuristic in SPARQL to get all types evaluated as exceptions.

```
SELECT distinct(?x) WHERE {
    ?x a <C> .
    ?x a ?t values (?t) {
        (<t1>) (<t2>) ... (<tn>)
    }
} LIMIT $limit OFFSET $offset
```

Fig. 5: Implementation of our optimized heuristic in SPARQL to get all exceptions (individuals) using the types found in Fig. 4

Likewise, we see that the ARIs of each axiom correspond to each other, giving an identical average ARI value ($\sim$ -0.1936) between the two figures. The previous work [13] suggests an ARI value $>$ 1/3 as an acceptance criterion for an axiom ϕ, to this effect, we observe that the number of accepted axioms is only 197 against 525 refused : 27,28% of acceptance for the dataset used. Among the axioms with an ARI = 1, we can found the following example : `SubClassOf(dbo:Chef dbo:Agent)` ; `SubClassOf(dbo:Venue dbo:Place)` ; `SubClassOf(dbo:RadioHost dbo:Person)` ; ...These candidate axioms are not present in the *DBPedia 3.9* ontology, so these axioms can be integrated into it and allow further inferences for the acquisition of new knowledge.

In a second step, we were interested in comparing the respective computation times between the 722 axioms. Figure 9 highlights the initial computation time observed for each axiom as a function of the computation time with our new implementation, so we obtain a ratio: If the ratio is less than 1, we observe a time saving with our optimisation. Otherwise, if this value is greater than 1 we observe a loss of time. We have chosen a logarithmic scale for the x-axis and y-axis in order to reduce the differences between the points and to improve the readability of the results.

We observe an average initial CPU time equivalent to 578 minutes against an average of 30 minutes with our

Algorithm 2 Test a `SubClassOf` axiom (optimisation) using `loop` operator from *Corese*

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

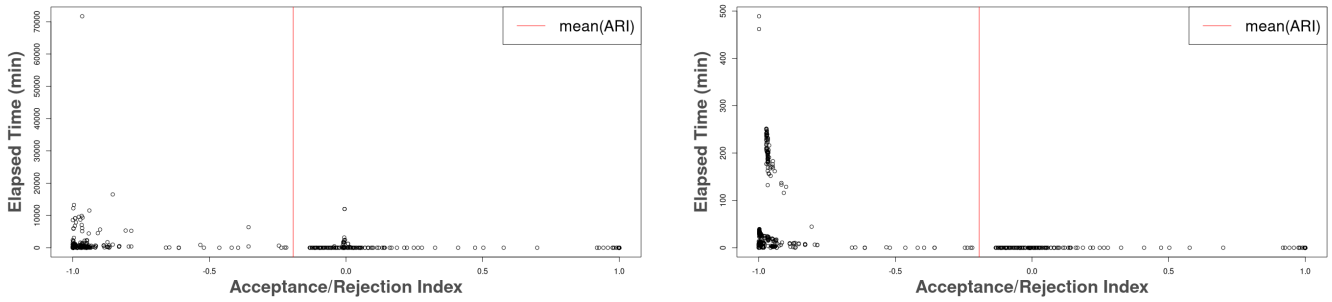
1: $limit \leftarrow 1000$
2: $url \leftarrow$ SPARQL Endpoint to query the chosen database.
3: compute the types t thanks to the request in Fig. 6 using url and $limit$.
4: $start \leftarrow 0$
5: $step \leftarrow 50$
6: $limit \leftarrow 10000$
7: **while** $start \neq len(types)$ **do**
8: $\quad end \leftarrow min(start + k, len(types))$
9: $\quad$ compute the instances e corresponding to request in Fig. 7 using url, t_n subset for $n \in [start, end]$ and $limit$
10: $\quad$ **if** $e \notin exceptions$ **then**
11: $\quad\quad exceptions$.insert(e)
12: $\quad$ **end if**
13: $\quad start \leftarrow start + min(len(types) - start, k)$
14: **end while**
15: $numExceptions \leftarrow len(exceptions)$

```
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: SPARQL request using `loop` operator from *Corese* : get all types evaluated as exceptions

```
SELECT distinct ?x WHERE {
    SERVICE <$url/sparql?loop=true&limit=$limit> {
        ?x a <C> .
        ?x a ?t values (?t) {
            (<t1>) (<t2>) ... (<tn>)
        }
    }
}
```

Fig. 7: SPARQL Request using `loop` operator from *Corese* : get all instances (using t results from SPARQL request presented in Fig.6) evaluated as exceptions.



(a) (b)

Fig. 8: Comparison of ARI values from the two approaches using a set of 722 axioms evaluated on DBPedia 3.9 with (a) our previous approach and (b) our optimized approach

optimisation. Similarly, our optimisation solves the problem of extremely long CPU computation times encountered for some axioms, as we can see in Figure 8a in which we have an axiom that cost 71,699 minutes to evaluate. Our optimisation significantly reduced the computational cost for this type of axiom to a maximum value of 489 minutes, a reduction by approximately 150. As a consequence, we have a major part of the axioms for which we observe a lower computation time: 593 axioms are faster to evaluate with our method, i.e. 82% of the candidate axioms tested. Nevertheless, we observe 129 axioms with an increase in computation time with our optimisation. However, the results reveal an average increase in computation time of about 57 minutes and a maximum loss of 244 minutes for this set of axioms. These are smaller losses compared to the gain we get from evaluating this set of candidate axioms.

This optimisation proposal avoids repetitive type computations, often computed several times for a given type (as explained in Section III). However, it is very

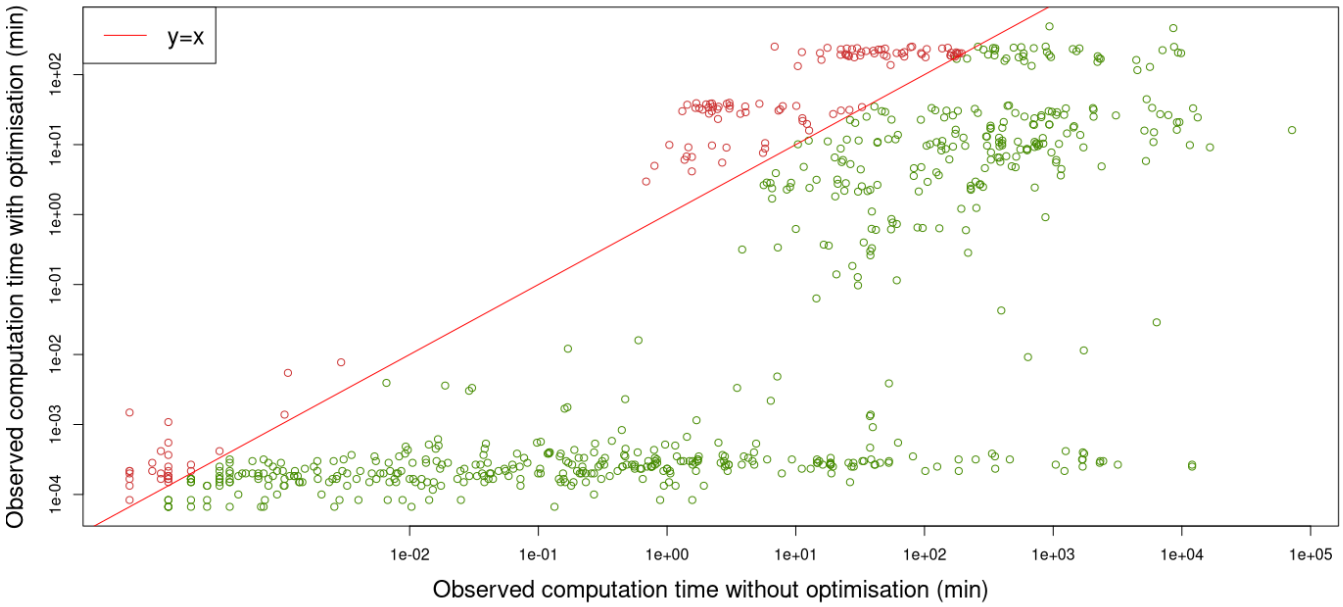


Fig. 9: Computation time of our previous method in function of our new one, with axioms for which we save time (in green) and the reverse (in red)

probable that for a given axiom ϕ, we do not observe type duplicates with the initial query (see Fig. 3). To this end, the initial query is logically faster as it allows the calculation of exceptions with only one SPARQL query, compared to at least two SPARQL queries with our optimisation. This explains the increase in computation time with our optimisation, however this increase is acceptable as it concerns a very specific set of axioms.

In the context of the contribution on Corese and the operator `loop`, we have chosen subsets of axioms that involve few (1 - 100,000), moderate (100,000 - 500,000) and many (+ 750,000) exceptions respectively. The objective is to analyse whether this new approach allows to obtain more advantageous CPU computation times while respecting the evaluation results found previously. The results presented in Table I highlight a progressive decrease in CPU computation time as the number of exceptions to be found increases, this suggests that the implementation of this operator optimises the truncation of queries internally making the computation cost less important than the first approach, in which we truncated the queries (see Algorithm 1).

V. Conclusion

We presented an optimisation of a possibilistic heuristic approach for the OWL 2 SubClassOf axioms assessment against RDF Data, able to decreasing the CPU computation time and offer new perspectives to this approach. This optimisation allows us to respect the original heuristic, as we observe no difference in the evaluation of axioms, while saving CPU time significantly. We propose two ways to implement this solution, a classical approach with SPARQL queries truncated by the user, making its application more tedious and a second approach where we put forward our expertise with the *Corese* tool and propose a SPARQL query allowing automatic truncation using a new operator, allowing at the same time to decrease the CPU computing time, which leads us to believe that this is the most interesting approach. Nevertheless, the evaluation of such an axiom still represents an important computation time as we have seen previously. We can assume that this optimisation will be able to be applied to the extension of our project, namely the extraction and evaluation of other axioms provided by OWL 2. One of the challenges is to analyse the generalisation of this optimisation to computational problems for SPARQL queries having more or less the same form, and thus bring the benefit of this 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. Pauwels, S. Zhang, and Y.-C. Lee, "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

TABLE I: Results of 9 axioms from our optimized approach compared to results obtained with the `loop` operator

Candidate subsumption axioms	Optimized approach ...		...using `loop` operator	
	`sum`(exceptions)	time (in ms)	`sum`(exceptions)	time (in ms)
SubClassOf (dbo:Genre dbo:Animal)	60	21	60	1,364
SubClassOf (schema:Product dbo:Agent)	41,707	23,968	41,707	5,953
SubClassOf (dbo:ArchitecturalStructure dbo:Hotel)	92,881	148,316	92,881	39,421
SubClassOf (dbo:ArchitecturalStructure schema:Park)	133,175	361,396	133,175	125,087
SubClassOf (schema:MusicAlbum dbo:Event)	148,238	361,590	148,238	172,319
SubClassOf (dbo:Organisation schema:RadioStation)	288,641	1,631,236	288,641	724,986
SubClassOf (schema:Person dbo:BasketballPlayer)	832,489	11,288,378	832,489	7,038,222
SubClassOf (schema:Person dbo:Automobile)	1,124,199	29,339,754	1,124,199	19,813,709
SubClassOf (schema:Person dbo:Infrastructure)	1,124,272	27,715,439	1,124,272	22,131,382

[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

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

[9] 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

[10] 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.

[11] 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

[12] F.-Z. C. Tettamanzi A.G.B. and F. 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. G. B. 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.

[14] R. Felin and A. G. B. 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

[15] 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.

[16] O. Corby, C. Faron-Zucker, and F. Gandon, "Ldscript: a linked data script language," in *International Semantic Web Conference*. Springer, 2017, pp. 208–224.