big data and cognitive computing

MDPI

Article

A Novel Algorithm for Multi-Criteria Ontology Merging through Iterative Update of RDF Graph

Mohammed Suleiman Mohammed Rudwan * and Jean Vincent Fonou-Dombeu

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Pietermaritzburg 3209, South Africa; fonoudombeuj@ukzn.ac.za
* Correspondence: m.suleiman.rudwan@gmail.com

Abstract: Ontology merging is an important task in ontology engineering to date. However, despite the efforts devoted to ontology merging, the incorporation of relevant features of ontologies such as axioms, individuals and annotations in the output ontologies remains challenging. Consequently, existing ontology-merging solutions produce new ontologies that do not include all the relevant semantic features from the candidate ontologies. To address these limitations, this paper proposes a novel algorithm for multi-criteria ontology merging that automatically builds a new ontology from candidate ontologies by iteratively updating an RDF graph in the memory. The proposed algorithm leverages state-of-the-art Natural Language Processing tools as well as a Machine Learning-based framework to assess the similarities and merge various criteria into the resulting output ontology. The key contribution of the proposed algorithm lies in its ability to merge relevant features from the candidate ontologies to build a more accurate, integrated and cohesive output ontology. The proposed algorithm is tested with five ontologies of different computing domains and evaluated in terms of its asymptotic behavior, quality and computational performance. The experimental results indicate that the proposed algorithm produces output ontologies that meet the integrity, accuracy and cohesion quality criteria better than related studies. This performance demonstrates the effectiveness and superior capabilities of the proposed algorithm. Furthermore, the proposed algorithm enables iterative in-memory update and building of the RDF graph of the resulting output ontology, which enhances the processing speed and improves the computational efficiency, making it an ideal solution for big data applications.

Keywords: ontology merging; ontology reuse; RDF graph; knowledge graph; ontology alignment; information integration

Citation: Rudwan, M.S.M.; Fonou-Dombeu, J.V. A Novel Algorithm for Multi-Criteria Ontology Merging through Iterative Update of RDF Graph. *Big Data Cogn. Comput.* **2024**, *8*, 19. https://doi.org/10.3390/bdcc8030019

Academic Editor: Miguel-Angel Sicilia

Received: 12 January 2024
Revised: 12 February 2024
Accepted: 17 February 2024
Published: 21 February 2024

1. Introduction

The proliferation of semantic web ontologies and their applications nowadays has made an abundance of information and knowledge available for reuse by a wide number of applications. In the era of big data, terabytes of data are being generated at each single moment via different types of media on the web. However, overlapping and inconsistency in the generated relevant data and information may exist. Therefore, there is a need to investigate and analyze such data for redundancy discovery, capturing new knowledge and building unified consistent knowledge bases that can be reused by respective applications. Thus, there is a massive need for techniques to merge knowledge from similar domain ontologies to produce up-to-date and integrated ontologies. Such a process is referred to as ontology merging [1,2]. Ontology merging aims to merge existing knowledge from heterogeneous sources to constitute a new one. However, despite the efforts devoted to ontology merging [3–9], the incorporation of axioms, individuals and annotations in the resulting ontologies remains challenging. Furthermore, certain studies [10] only relied on lexical analysis of ontology concepts to perform the merging, which does not cover the semantic analysis of features of the candidate ontologies. Consequently, existing ontology-merging

solutions produce new ontologies that do not include all the related and relevant semantic features from the candidate ontologies. Furthermore, such output ontologies should meet the required quality standard to ensure their usefulness [11–14]. Quality ontology merging requires knowledge in the output ontology to be complete, has a minimum amount of knowledge redundancy, has a high level of connectivity and acclivity, is concise and consistent, and enables inferences with constraints [12–14]. To address the limitations of existing ontology-merging solutions, this paper proposes a novel algorithm for multi-criteria ontology merging that automatically builds a new ontology from candidate ontologies by iteratively updating its RDF graph in the memory. The algorithm begins by extracting the concepts, logical axioms and individuals from both the base and candidate ontologies. Thereafter, the concepts and annotations are aligned and merged to build an RDF graph, which serves as the input for the subsequent stage. Next, logical axioms extracted from the candidate ontologies are matched against all the logical axioms within the base ontology. The decision to include or exclude these axioms in the output ontology is guided by a predefined similarity score threshold. The updated RDF graph serves as the input for the final stage, where individuals are matched and merged to construct the final RDF graph of the resulting output ontology. The proposed algorithm uses a similarity-based framework to assess the concept similarities to guide the subsequent merging processes. The proposed algorithm leverages state-of-the-art Natural Language Processing tools such as fuzzy string-matching algorithms, Word2Vec, BERT and WordNet as well as a Machine Learning-based framework, namely, SM-DTR, to assess the similarities and merge various criteria. The key contribution of the proposed algorithm lies in its ability to merge relevant features from candidate ontologies, such as logical and direct/declarative axioms, annotations, individuals and hierarchical structure, to build a more accurate, integrated and cohesive output ontology. The proposed algorithm was tested with five ontologies of different computing domains, which were downloaded from various repositories on the web, and evaluated in terms of its asymptotic behavior, quality and computational performance. The analysis of the experimental results indicated that the proposed algorithm produced output ontologies that met the integrity, accuracy and cohesion quality criteria better than related studies. This performance demonstrated the effectiveness and superior capabilities of the proposed algorithm for ontology merging. Furthermore, the proposed algorithm enabled iterative in-memory updating and building of the RDF graph of the resulting output ontology, which enhanced the processing speed and improved the computational efficiency, making it an ideal solution for big data applications.

The main contribution of this paper can be summarized in the following points:

- The proposed algorithm takes into account multiple criteria in the ontology-merging process, including logical and direct/declarative axioms, hierarchal structure, individuals, and annotations. This results in high-quality output ontologies that are integrated, accurate and cohesive.
- We introduce a multi-level measure of similarity between the matched components and vocabulary in the ontologies. This measure guides the decision-making process concerning how to seamlessly incorporate relevant vocabulary and knowledge from the candidate ontologies into the output ontology.
- The proposed algorithm leverages the in-memory RDF graphs mechanism, enabling efficient processing capability for periodic and continuous updates and smooth ontology merging. This computational advantage holds value in various settings, especially for large-scale applications dealing with frequent data generation, including big data.

The remainder of this paper is organized as follows. Section 2 delves into the existing literature and related work, providing valuable context for our research. In Section 3, we detail the materials and methods used in the proposed algorithm. Section 4 presents the proposed algorithm. In Section 5, we present our experimental results and engage in a comprehensive discussion of their implications. In Section 6, we compare our results and evaluate them against existing works. Finally, in Section 7, we draw a conclusion and offer insights into potential directions for future research endeavors.

2. Literature Review

Previous endeavors have been devoted to providing guidelines for quality ontology merging. The authors of [12] proposed a comprehensive framework for merging candidate ontologies into output ontologies that meet the integrity and cohesion quality criteria. The integrity criterion advocates for comprehensive knowledge coverage and minimal redundancy and emphasizes that the output ontology from the merging encapsulates as much pertinent knowledge as possible while trimming excess repetition, whereas the cohesion criterion indicates how related the properties within the ontology are. Another study [13] defined four quality criteria of ontologies, including accuracy, understandability, cohesion, and conciseness, as well as the mathematical formulations for each of these criteria. In [15], ontology quality metrics were classified into two distinct groups, namely, schema metrics and knowledge base metrics. The schema metrics assess the design of the ontology and its knowledge representation capability, while the knowledge base metrics examine the incorporation of instance data into the ontology and how effectively it leverages the knowledge outlined in the schema.

In [16], the authors proposed a novel method for merging domain ontologies utilizing granular computing. The method consists of four major operations, including association, isolation, purification, and reduction. The method works by comparing the concepts of ontologies, aiming to reduce the level of granularity during the merging process. This study considered the labels of classes and the taxonomy, while other criteria, including properties, logical axioms, and individuals, were not considered in the proposed method. In another study [17], the authors developed an algorithm called ATOM for taxonomy merging. They harnessed the is-a and equivalent classes relationships to match different concepts. Their focus was on taxonomy and individuals, while other criteria that can enrich the output ontology, such as direct and logical axioms, as well as object properties were not considered. In [18], the researchers proposed a semi-automatic method for merging large ontologies using graphDB for big data processing. Their method is well suited for modularization-type problems such as the importing of specific module or part of the ontology into others. However, a main shortcoming of their algorithm is that it exploits the entire sub-classes of the matched entities into the resulting output ontology, assuming the relevance of the whole candidate ontologies' sub-class taxonomies. This does not comply with the reduction and cohesion guidelines prescribed in [12–14]. Additionally, the merging process involves manual intervention of human operators, which may slow down the process. These shortcomings are well addressed in the fully automated algorithm proposed in this study. In another work [11], the authors proposed a semi-automatic framework for merging smart factories ontologies. Their methodology includes three major tasks, namely, preprocessing, matching, and post-processing. They performed several operations embedded within those tasks, including spell-checking of the concepts' labels, tokenization and translation, structure analysis and user confirmation. The inclusion of similar concepts is based on two threshold values that determine the relevance to minimize the rate of rejection. However, their method does not process annotations, which are relevant features of ontologies. This shortcoming is overcome in our proposed algorithm in this study. In [3], a method called the Hybrid Semantic Similarity Measure (HSSM) was proposed for merging ontologies. The aim was to remove redundancy and improve storage efficiency, which are important aspects of quality ontology merging. The method was developed using Formal Concept Analysis (FCA) and semantic similarity measures. Although their method is effective, it does not cover other relevant ontology elements, such as logical axioms and annotations. Many studies [15,19–23] utilized the lexical database WordNet in their ontology-merging methods due to their effectiveness in the semantic analysis of terminologies. Our proposed algorithm also utilized WordNet for synonym extraction purposes.

To the best of our knowledge, the majority of previous studies [8–10,24] did not process the logical axioms and annotations in their ontology-merging methods. This shortcoming is addressed in our proposed algorithm through the processing of various

criteria, including logical axioms, individuals and annotations, with the aim of producing quality output ontologies that include relevant features from candidate ontologies as well as meeting the prescribed ontology-merging quality criteria [12–14]. A comparative and exhaustive analysis of 19 studies was undertaken to review all the criteria used within their respective methodologies for ontology merging. The findings of this comparative analysis are presented in Table 1. To represent the extent to which a criterion has been fulfilled in each study, we used the following encodings in Table 1: (1) ✔ indicates that the given criterion is met comprehensively; (2) ✗ signifies the nonfulfillment of the criterion at all; and (3) ≈ denotes an intermediate degree of satisfaction, that is, a partial fulfillment of the criterion.

Table 1. Criteria for ontology merging in the literature.

#	Study	Year of Study	Direct Axioms (Declarative Axioms)				Taxonomy	Logical Axioms	Annotations
			Properties	Classes		Individuals			
				Lexical	Semantic				
1	[16]	2021	✗	✔	✔	✗	✔	✗	✗
2	[3]	2020	✗	✔	✔	✗	✔	✗	✗
3	[11]	2022	✗	✔	✔	✗	✔	✔	✗
4	[7]	2021	✗	✔	≈	✗	✗	✗	✗
5	[6]	2020	✗	✔	✗	✗	✗	≈	✗
6	[25]	2020	✗	✔	≈	✗	✔	≈	✗
7	[19]	2010	✗	✔	✗	✔	✔	✗	✗
8	[19]	2010	✔	✔	✔	✗	✗	✗	✗
9	[15]	2019	✗	✔	✔	✗	✔	✗	✗
10	[8]	2022	✗	✔	✗	✗	✔	✗	✗
11	[10]	2021	✔	✔	✗	✗	✗	✗	✗
12	[20]	2012	✗	✔	✔	✗	✗	✗	✗
13	[21]	2021	✔	✔	✗	✗	✗	✗	✗
14	[22]	2021	✔	✔	✔	✗	✔	✗	✗
15	[9]	2023	✗	✔	✔	✗	✔	✗	✗
16	[2]	2022	✗	✔	✔	✗	✔	✗	✗
17	[1]	2022	✔	✔	✗	✗	✔	✗	✗
18	[23]	2022	✔	✔	✗	✗	✔	✗	✗
19	[18]	2014	≈	✗	✗	✔	✔	✗	✗
20	Proposed method	2023	✔ (Embedded in logical axioms alignment)	✔	✔	✔	✔	✔	✔

Table 1 offers an insightful glimpse into the spectrum of criteria governing ontology merging, along with the extent to which the corresponding studies have fulfilled them. It is obvious that a substantial portion of the studies investigated did not consider the logical axioms in the merging process. Additionally, none of the previous studies considered the annotations, while few have integrated the individuals within their methodologies. Another noticeable finding is that no study has endeavored to tackle the entirety of these criteria collectively. As a result, the resulting output ontologies from existing ontology-merging solutions do not satisfy the prescribed quality criteria of integration, cohesion and completeness.

Ontology merging is a complex and demanding task due to the heterogeneity in the structures and semantics of the ontologies being merged. Despite the advancements witnessed in the recent literature, where various solutions for merging ontologies have been proposed [2,7,11,15], it is noteworthy that many of the previous studies do not cover the holistic dimension of ontology merging. In other words, the majority of previous studies do not merge important features of ontology, including logical axioms, instances and annotations, in the output ontologies [9,15,16]. Based on the analysis of the shortcomings above, the proposed algorithm in this work covers the full spectrum of the criteria shown

in Table 1 in the merging process. This ensures that the majority of the quality criteria prescribed [12–14] are met by the resulting output ontologies from the proposed algorithm.

3. Materials and Methods

This section presents the Natural Language Processing and Machine Learning techniques used as well as the methods employed to evaluate the proposed algorithm.

3.1. Natural Language Processing Tools

Natural Language Processing (NLP) enables the analysis of human natural language text to extract meaningful insights utilizing different computer algorithms and techniques [26]. In this work, state-of-the-art NLP techniques were utilized, including Fuzzy String-matching algorithms, Word2Vec and the WordNet lexical database.

3.1.1. Fuzzy String-Matching Algorithms

Fuzzy string-matching algorithms are algorithms that rely on fuzzy logic where the uncertainty of the final answer to a problem is given. Usually, the fuzzy generates a result in terms of a decimal number ranging between 0 and 1, inclusive. There are several fuzzy string-matching algorithms for string similarity detection to explain the degree of similarity between pairs of strings. These include, but are not limited to, Jaro–Winkler, Jaccard, Levenshtein, Longest Common Subsequence (LCS), Term Frequency-Inverse Document Frequency (TF-IDF), N-gram, and many more [26,27]. In our proposed algorithm, we have utilized both Jaccard and Jaro–Winkler in the merging process. Jaccard was used for individuals matching, while Jaro–Winkler was utilized in the logical axioms alignment. The two algorithms were chose because they are efficient in the syntactic matching of strings that contain two or more labels [26]. Specifically, the Jaro–Winkler algorithm works well in processing axioms expressions that contain multiple tokens/words [26]. The Jaccard similarity coefficient algorithm was originally designed for set theory applications [27] but has also been adapted to assess the similarity between strings. This algorithm assesses the resemblance of two strings by analyzing their individual characters and identifying shared characters to ascertain their degree of similarity. The Jaro–Winkler algorithm is based on the Jaro distance. It measures the similarity between two strings by counting the common characters and transpositions while giving extra weight to the common prefixes. The details of the mathematical representation of how both the Jaccard and Jaro–Winkler algorithms calculate the similarity scores of strings can be found in [27].

3.1.2. Word2Vec

Word2Vec is an artificial neural network-based approach that aims to convert words in natural language into vectors [28]. It has two variants, namely, Continuous Bag-of-Words (CBOW), and Skip-gram, each of which works for different purposes. Word2Vec was used in our experiment at the logical axioms merging stage, wherein logical axioms of the base ontology were extracted first and then each axiom expression was tokenized by splitting the string into word components based on space, brackets and commas delimiters. Thereafter, these tokens were trained via Word2Vec, which resulted in a trained model. The output model was then used to detect similarities in the input axioms from candidate ontologies with those in the base ontology.

3.1.3. WordNet

WordNet is a huge lexical database of the English language. It consists of all the English language vocabulary, definitions, synonyms, examples, and much more. It was first developed at Princeton University in 1995 [29]. We have used WordNet at the concepts and annotations merging stage in the proposed algorithm. It was used to obtain the synonyms of similar concepts and add them to the output ontology for enrichment. The reliance on WordNet was due to its accuracy and effectiveness in defining concepts, as attested by experts [22].

3.2. Machine Learning Techniques

In this work, we have utilized Bidirectional Encoder Representations from Transformers (BERT) [30], a deep learning pre-trained model developed by Google. BERT is trained in a bidirectional manner on an unlabeled text to generate output text in response to input queries/questions from the user in the form of text. It analyzes pre- and post-text to figure out the its semantic meaning, and then it generates the new text for the user. The fine-tuning feature gives the user extra option to customize the model to a specific purpose by retraining it with the new features for a specific use case [30]. BERT can be used for many natural language processing tasks, including classification of text, detection of similarities, and much more. We have also used a Machine Learning-based framework, namely, the Similarity Model with Decision Tree Regression (SM-DTR), in the proposed algorithm. BERT was harnessed for merging annotations, while SM-DTR was used for concepts and classes merging in our proposed algorithm.

SM-DTR is an ontology alignment method that detects the similarities between ontologies at both the lexical and semantic levels [26]. It consists of three main components, namely, fuzzy string-matching, BERT pre-trained model, and Decision Tree Regression (DTR) classifier. Each of the fuzzy string-matching algorithms used in the model, and BERT, calculates the similarity scores between concepts from two ontologies. Then, the output is fed into the decision tree regression classifier for calculating the final similarity score. The SM-DTR method is presented in detail in [26].

3.3. Evaluation of the Proposed Algorithm

This section presents the various methods used to evaluate the proposed algorithm, including asymptotic analysis and assessment of the quality of the output ontology.

3.3.1. Asymptotic Analysis

Asymptotic analysis aims to analyze algorithms' efficiency in terms of the time complexity and space complexity. Our interest in this study is in the time complexity only. In time complexity, given n as the number of inputs, we analyze the algorithm and obtain the worst case scenario using the big O notation. For instance, $O(n^2)$ indicates that the time complexity of the algorithm analyzed is exponential, where the growth rate doubles with additional inputs. In addition to that, we have also measured the execution time of the proposed algorithm for merging each of the candidate ontologies with the base ontology.

3.3.2. Quality Analysis of Output Ontology

Assessing the quality of the output ontology involves the quantitative analysis of the output ontology's vocabulary, such as the number of classes, object properties, axioms, etc. In this study, a number of graph metrics that assess the complexity of the design of ontology were used to evaluate the quality of the output ontology from the proposed algorithm. The graph metrics considered include the Average Depth (AD), Maximum Depth (MD), Average Breadth (AB), Maximum Breadth (MB), Absolute Root Cardinality (ARC), and Absolute Leaf Cardinality (ALC). The AD, MD, AB, and MB metrics serve as indicators of the ontology accuracy, presenting how well the ontology aligns with the domain it models. The ARC and ALC metrics provide insights into the ontology's cohesion, revealing the extent to which classes are interlinked and how instances are distributed among them [13,17]. The mathematical formulas for the calculations of these graph metrics can be found in [13].

4. Proposed RDF-MOM Algorithm

The proposed algorithm works at three main stages, namely, merging concepts and annotations, merging logical axioms and merging individuals. The algorithm begins by extracting ontologies' concepts, logical axioms and individuals and saves them into text files. This process is performed for both the candidate ontologies denoted O_n, where n is the index of the candidate ontology, and the base ontology, denoted O_{base}. Next, the

concepts and annotations are aligned and subsequently merged. The outcome is an RDF graph, which serves as the input for the subsequent stage. In the second stage, the logical axioms extracted from the candidate ontologies are matched against all the logical axioms within the base ontology. The decision to incorporate or exclude these axioms hinges on a predefined similarity score threshold. In the final stage, individuals are matched and merged to build the final RDF graph of the output ontology. The proposed algorithm is presented in detail next.

The proposed algorithm for merging ontologies through iterative updating of the RDF graph is presented in Algorithm 1. Algorithm 1 initiates the process of aligning the concepts and annotations in line 3, where Algorithm 2 is invoked. The output of Algorithm 2 is an RDF graph, which serves as the input for the subsequent task of merging the logical axioms. The candidate ontology is converted into an RDF graph in line 4. Line 5 initiates an array that contains the list of RDF and OWL keywords that are going to be used in a later stage. In lines 6 and 7, the logical axioms are extracted using Algorithm 3. This is followed by the creation of word embeddings for the axioms in the base ontology in line 8. Because the number of logical expressions is needed for calculation of the average of similarity score later, the *sum* and *count* variables are declared in line 9. Starting from line 11, the algorithm iterates through the logical axioms in the candidate ontology in both their URI and label formats to align them with their counterparts in the base ontology. In lines 12–13, each logical axiom expression is tokenized to acquire the most relevant vector for each token. Tokens are words that compose a logical axiom expression. In lines 19–20, each token from the candidate ontology is tested against other vectors using the *boW2VModel* created in line 8. It is essential to note that Word2Vec models cannot recognize previously unseen vocabulary when initially constructed. Hence, the algorithm keeps track of the tokens discovered by the model and those that were not in lines 14–15 and lines 24 and 26. Once all the tokens have been processed through the *boW2VModel*, it is checked if they are all represented (line 27). If so, the average similarity score is calculated by dividing the *sum* by the *count* in line 28. The next step involves testing this average value. If it is greater than 0.50 and less than the predefined *logAxi_thr* threshold, the axiom is added to the RDF graph in lines 29–30. Otherwise, the algorithm proceeds to iterate again over the next axioms in the candidate ontology. If not all the tokens are represented in *boW2VModel*, the algorithm proceeds to line 31. From lines 32–42, tokens that were not represented are matched with all the logical axioms in the base ontology with the *bLab* and *bUri* text files using the Jaro–Winkler fuzzy string-matching algorithm (line 37). Subsequently, the average similarity score for all the tokens is calculated in line 40 and it is assessed whether the average similarity score meets the *logAxi_thr* threshold or is below 0.50 in line 41. If the condition holds, the axiom is discarded; otherwise, it is integrated into the RDF graph (lines 41–42).

The algorithm proceeds with the merging of individuals in line 43 with Algorithm 6.

Algorithm 2 is used in line 3 of the proposed algorithm (Algorithm 1) to merge the concepts and annotations into the output ontology. The algorithm takes as inputs in line 1 the URIs of the concepts and annotations for the base and candidate ontologies, *boUrl* and *coUrl*, respectively, the concepts similarity score threshold *conAx_thr,* the concepts weight *concWght*, and the annotations weight *annWght*. Thereafter, the graph structures of the base and candidate ontologies are extracted in lines 3–4. In line 5, the algorithm iterates through the concepts and their respective URIs within the candidate ontology to explore the relationships between them and their counterparts in the base ontology. Within the loop in line 5, an inner loop in line 12 traverses all the concepts within the base ontology as well as their associated annotations. These annotations are compared with the current concept's annotations in the candidate ontology using the SM-DTR method in lines 12–17. Additionally, the annotations in the candidate ontology are matched to those of the base ontology using BERT in lines 23–28. From these iterative processes, the algorithm identifies the most similar concepts and annotations, weighed *concWght* and *annAxiWght*, respectively. These results are subsequently aggregated in line 29, and the cumulative

similarity score is evaluated against the predefined *conAx_thr* similarity threshold in line 30. If this condition is met, signifying that the concepts share substantial commonality, their synonyms are extracted from WordNet and added to the in-memory RDF graph in lines 30–36. If the condition in line 30 is not met, a second check is performed in line 37 to determine whether the similarity score is higher than 0.50, that is, there is some degree of similarity. In such a case, the concept from the candidate ontology is added to the graph in line 38. If neither of the conditions in lines 30 and 37 is true, then the concept is discarded and the outer loop in line 5 proceeds to iterate over the remaining concepts and annotations within the candidate ontology.

Algorithm 1: RDF-MOM Algorithm

```
1  Inputs: boUrl, coUrl, conAx_thr, logAxi_thr, concWght, dirAxiWght
2  Outputs: myNewGraph
3  myNewGraph ← conceptsAndAnnotationsMerging(boUrl, coUrl, conAx_thr, concWght, dirAxiWght)
4  coGraph ←  toRDFGraph (COurl)
5  owlLogAx_keywords ← [subClassOf, ObjectUnionOf, . . .]
6  boLogAxLab, boLogAxUris ← logicalAxiomExtraction (boUrl)
7  coLogAxLab, coLogAxUris ← logicalAxiomExtraction (coUrl)
8  boW2VModel ← W2Vec.trainModel(boLogAxLab)
9  sum, count ← 0
10
11 for conLab, conUri in coLogAxLab, coLogAxUris do:
12       temp_list1 ← sentenceToList(coLogAxLab)
13       temp_list1_uris ← sentenceToList(coLogAxUris)
14       unseen_words[] ← null
15       seen_words[] ← null
16
17        counter, avg_sum, max_sim ← 0
18       max_lab ← null
19       for word in conLab do:
20            max_sim, max_lab ← GetMaxSimToken(boW2VModel.predictSimilarity(word))
21       if similar token from conLab was found, then:
22            counter ← counter+1
23             sum ← sum + max_sim
24            seen_words.append([word, conLab, conUri, max_sim])
25       else:
26            unseen_words.append[congLab, conUri]
27       if length of unseen_words == 0, then:
28             avg ← sum/count
29            if avg <logAxi_thr and avg >0.50, then:
30                 myNewGraph← addAxiomToRDFGraph(myNewGraph,conLab, ConUri)
31       else:
32            for unseenWord in unseen_words do:
33                 max_val ← 0
34                 max_lab, max_uri ← null
35                 for  bLab, bUri in boLogAxLab, boLogAxUris do:
36                      for bLabWord, bUriToken in bLab, bUri:
37                           max_val,max_lab,max_uri ← GetMaxSimToken(JaroWinklerSim(unseen_words [0], bLabWord))
38                 sum ← sum + max_val
39                 count ← count + 1
40            avg ← sum/count
41            if avg < logAxi_thr and avg > 0.50, then:
42                 myNewGraph. addAxiomToRDFGraph(myNewGraph, conLab, ConUri)
43 myNewGraph ← mergeIndividuals(myNewGraph, COurl)
44 return myNewGraph
```

Algorithm 2: conceptsAndAnnotationsMerging ()

```
1  Inputs: boUrl, coUrl, conAx_thr, concWght, annWght
2  Outputs: new_rdfGraph
3  newRdfGraph ← toRDFGraph(boUrl)
4  cand_grpah ← toRDFGraph(coUrl)
5  for cand_conLab, cand_iri  in cand_graph do:
6        if cand_conLab is empty, then:
7              continue
8         else:
9              candAnn ← getDecAnnotations(cand_iri)
10             maxSimCon ← ""
11             maxSimVal ← 0
12             for base_conLab, base_iri in newRdfGraph do:
13                   if base_conLab is NOT empty, then:
14                         s1 ← SM_DTR_SIMILARITY(cand_conLab, base_conLab)
15                         if s1 > maxSimVal then:
16                               maxSimVal ← s1
17                               maxSimCon ← base_iri
18             sim1 ← maxSimVal
19             sim2 ← 0
20             baseAnn ← getDecAnnotations(maxSimCon)
21             candAnn ← getDecAnnotations(cand_iri)
22             common_Ann ← candAnn ∩ baseAnn
23             if common_Ann is NOT empty, then:
24                   for ann in common_Ann do:
25                         baseAnn ← getAnnVal(base_iri, baseDirAxAnn)
26                         candAnn ← getAnnVal(cand_iri, candAnn)
27                         annSimilarity ← BERT(baseAnn, candAnn)
28                         sim2 ← annSimilarity
29             overall_sim ← (sim1 * concWght) + (sim2 * annWght)
30             if overall_sim > = conAx_thr then:
31                   synSet[]←WordNet.synSet(cand_conLab) ∩ WordNet.synSet(base_conLab)
32                   if cand_conLab is NOT exact match with base_conLab then:
33                         synSet.addElement(cand_conLab)
34                   if synSet is NOT empty, then:
35                         newRdfGraph.addTriple((base_iri, nameSpace.synonyms, synSet))
36                   newRdfGraph.addTriple((base_iri, nameSpace.common_Ann, candAnn))
37             else if overall_sim > 0.50, then:
38                   newRdfGraph.addNewClass((cand_iri, CLASS))
39             else:
40                   continue
41 return newRdfGraph
```

Algorithm 3 is used in lines 6–7 of the proposed algorithm (Algorithm 1) to extract all the logical axioms from a given input ontology. The algorithm extracts the axioms of the input ontology in line 3. In lines 4–5, the algorithm proceeds to create two text files to store the URIs and labels of the axioms. Subsequently, in lines 6–8, the URIs extracted in line 3 are stored into the *logAxioms_URIs* text file, while the labels are stored in the *logAxioms_Labels* text files in lines 9–12.

Algorithm 4 is used in lines 30 and 42 of the proposed algorithm (Algorithm 1) to recursively add axioms to the RDF graph of the output ontology in the memory. The algorithm addresses two scenarios. The base case in lines 5–7 is when the axiom comprises precisely three tokens of keywords, URIs of vocabulary or values. Consequently, these tokens are added to the graph in lines 6–7 of Algorithm 4. However, when the axiom contains more than three tokens, the recursive case in lines 8–10 is executed. Upon the completion of the recursive process, the algorithm returns the updated graph *newGraph*.

Algorithm 3: logicalAxiomExtraction ()

```
1  Inputs: ontURL
2  Outputs: logAxioms_URIs, logAxioms_Labels
3  Axioms_URIs ← Extract_OWLLogAxioms(ontURL)
4  logAxioms_URIs ← CreatFile('ontology_name_URIs')
5  logAxioms_Labels ← CreatFile('ontology_name_Labels')
6  for axiom in Axioms_URIs do:
7  logAxioms_URIs.writeLine(axiom)
8  logAxioms_URIs.close()
9  for line in logAxioms_URIs do:
10 axiomWithLabels ← Tokenize line and get embedded labels from URIs using SPARQL
11 logAxioms_Labels.writeLine(axiomWithLabels)
12 logAxioms_Label.close()
13 return logAxioms_URIs, logAxioms_Labels
```

Algorithm 4: addAxiom()

```
1  Inputs: inputGraph, axiomLabExp[], axiomUriExp[]
2  Outputs: newGraph
3  newGraph ← inputGraph
4  x = isAxiomKeyword (axiomLabEx [0])
5  if length(axiomLabExp) == 3, then:
6        newGraph.addTriple((axiomUriExp [1], x [0], axiomUriExp [1]))
7        newGraph.addTriple((axiomUriExp [1], RDFS.comment, "new merged axiom!"))
8  else:
9        if x [1] != "concept_label", then:
10             newGrph=addAxiom(newGraph,axiomLabExp [1:endOfList],axiomUriExp [1:endOfList])
11 return newGraph
```

Algorithm 5 is used in line 4 of Algorithm 4 to accept a term as an input and return a Boolean value. It tests whether the input term is a reserved RDF or OWL2 keyword or not. If the term is in the list *owlLogAx_keywords* created in line 5 of Algorithm 1, then it returns true (lines 5–6), and it returns false otherwise (lines 7–8).

Algorithm 5: isAxiomKeyword()

```
1  Inputs: term
2  Outputs: result[]
3  result[] ← null
4  index = owlLogAx_keywords.indexOf(term)
5  if index >= 0, then:
6        result ← [ owlLogAx_keywords[index]]
7  else:
8        result ← [ term, "concept label"]
9  return result
```

Algorithm 6 is used in line 43 of the proposed algorithm (Algorithm 1) to perform the merging of individuals. It accepts as inputs the graphs of the base and candidate ontology, along with a predefined threshold for the individual similarity scores (line 1). A SPARQL query in lines 3–7 is executed in lines 8 and 9 to retrieve all the individuals in the base and candidate ontologies. A loop in line 11 is used to process all the individuals in the candidate ontology. Within the loop in line 11, another loop is nested to iterate over all the individuals in the base ontology. The inner loop employs the Jaccard algorithm to assess the similarity score between the individuals in lines 15–20. Next, the maximum similarity score is tested against the threshold *threshold* in line 21. If the maximum similarity score achieved the *threshold*, then the two concepts are similar and no change is made in the graph. Otherwise, if the similarity score is greater than 0.50, the individual is added to the

graph in line 22. If not, the individual is discarded. The updated *newGraph* is then returned in line 23. The *newGraph* is the final graph of the merging process and represents the graph of the output ontology in line 43 of the main algorithm (Algorithm 1).

Algorithm 6: mergeIndividuals()

```
1  Inputs: basetGraph, candGraph, threshold
2  Outputs: newGraph
3  query ← " SELECT ?individual_uri ?label
4            WHERE {
5            ?individual rdf:type ?classs.
6            ?individual rdfs:label ?label.
7          }
8  baseResults ← baseGraph.executeQuery (query)
9  candResults ← candGraph.executeQuery (query)
10 newGraph ← baseGraph
11 for row1 in candResults do:
12       max_val ← 0
13       max_label ← null
14       max_uri ← null
15       for row2 in baseResults do:
16             jacard_sim ← jaccardSimilarity(row1.label, row2.label)
17             if jaccard_sim > max_val, then:
18                   max_val ← jaccard_sim
19                   max_label ← row2.label
20                   max_uri ← rwo2.individual_uri
21       if max_val < threshold and max_val > 0.80  then:
22             newGraph.addIndividual (row1)
23 return newGraph
```

5. Experimental Results and Discussion

5.1. Dataset

The dataset in this study constituted five ontologies of various computing domains, which were obtained from different repositories over the web [31–34]. These ontologies included the Code Ontology, Software Ontology, Game Ontology, Deep Learning Ontology and Internet of Things Ontology. The Code Ontology was considered the base ontology and encoded *Obase* and the remaining four ontologies, namely, Software Ontology, Game Ontology, Deep Learning Ontology and Internet of Things Ontology, were the candidate ontologies and encoded *Ocandidate-1*, *Ocandidate-2*, *Ocandidate-3*, and *Ocandiadte-4*, respectively.

5.2. Experimental Setup

The Python and Java programing languages were utilized in the experiment. Java was used to extract concepts and axioms from the ontologies and save them into text files. Python was used to implement the proposed algorithm. The preprocessing and manipulation of the RDF graph for the merging of the ontologies were undertaken using APIs, including Jena API, owlready2, and rdflib. The visualization of the ontologies was performed with the online tool WebVowel version 1.1.7 [35]. Two tools were utilized to obtain and compare the quality metrics of the ontologies, namely, OntoMetrics [36] and Protégé [37]. OntoMetrics is an online tool that enables the automatic calculation of the base metrics, knowledge base metrics, and graph metrics of an ontology. Additionally, the Pallet and Hermit reasoners were used within Protégé to validate the consistency of the output ontologies.

5.3. Analysis of Base Metrics of Ontologies

The proposed algorithm was tested with five ontologies, including a base ontology *Obase*, whose structure was preserved, and four candidate ontologies to be merged with

the base one. The base metrics of the base and candidate ontologies were obtained and analyzed. The base metrics included the number of classes, individuals, data properties, object properties, logical axioms, and all axioms. For each candidate ontology, the proposed algorithm was applied to merge it with the base ontology. It is important to indicate that the proposed algorithm merged the base ontology and a candidate ontology at a time. Subsequently, the analysis of the base metrics for each output ontology was undertaken. Each output ontology resulting from the matching of a candidate ontology with the base ontology is encoded *Ooutput-i*, where $1 \leq i \leq 4$. Tables 2–5 present the based metrics for each triple <base, candidate, output> ontology.

Table 2. Base metrics of the *Obase*, *Ocandidate-1* and *Ooutput-1* ontologies.

Ontology	# of Classes	# of Logical Axioms	# of Individuals	# of Data Properties	# of Object Properties	TOTAL # of All Axioms
Obase	65	548	17	13	94	1097
Ocandidate-1	1845	5378	119	5	43	15,109
Ooutput-1	71	554	17	13	94	1115

Table 3. Base metrics of the *Obase*, *Ocandidate-2* and *Ooutput-2* ontologies.

Ontology	# of Classes	# of Logical Axioms	# of Individuals	# of Data Properties	# of Object Properties	TOTAL # of All Axioms
Obase	65	548	17	13	94	1097
Ocandidate-2	37	111	0	6	33	334
Ooutput-2	97	580	17	13	94	1193

Table 4. Base metrics of the *Obase*, *Ocandidate-3* and *Ooutput-3* ontologies.

Ontology	# of Classes	# of Logical Axioms	# of Individuals	# of Data Properties	# of Object Properties	TOTAL # of All Axioms
Obase	65	548	17	13	94	1097
Ocandidate-3	61	60	0	0	0	227
Ooutput-3	65	548	17	13	94	1097

Table 5. Base metrics the *Obase*, *Ocandidate-4* and *Ooutput-4* ontologies.

Ontology	# of Classes	# of Logical Axioms	# of Individuals	# of Data Properties	# of Object Properties	TOTAL # of All Axioms
Obase	65	548	17	13	94	1097
Ocandidate-4	21	60	0	13	14	274
Ooutput-4	65	548	17	13	94	1097

In Tables 4 and 5, it can be noticed that the base metrics for the base and output ontologies are the same. This indicates that the candidate ontologies O_3 and O_4 have no similarities/overlaps with the base ontology. In contrast, in Tables 2 and 3, the base metrics for the output ontologies are slightly higher than those of the base ontology, apart from the number of individuals, data properties, and object properties, which remained the same as those of the base ontology. This indicates that the candidate ontologies *Ocandidate-*$_1$ and *Ocandidate-*$_2$ have some similarities with the base ontology.

It is shown in Table 2 that the number of classes in the *Obase* and *Ooutput-1* ontologies is 65 and 71, respectively. This indicates that six concepts of the *Ocandidate-1* ontology were found to be similar to some concepts in the *Obase* ontology and were merged into the *Ooutput-1* output ontology. The same can be observed in Table 3, where 32 concepts of the *Ocandidate-2* ontology met the similarity test and were merged into the *Ooutput-2* ontology. Furthermore, the rightmost column of Table 3 indicates a significance difference of

104 between the total number of axioms in the *Obase* and *Ooutput-2* ontologies. This indicates that 104 axioms of the *Ocandidate-2* ontology met the similarity tests and were added or merged into the *Ooutput-2*. These findings revealed that the *Ocandidate-2* ontology had the most substantial overlaps with the *Obase* ontology, followed closely by the *Ocandidate-1* ontology. In contrast, the *Ocandidate-3* and *Ocandidate-4* ontologies exhibited no significant overlaps or similarities with the *Obase* ontology, as shown in Tables 4 and 5, respectively.

The proposed algorithm matched each candidate ontology to the *Obase* base ontology and iteratively updated the RDF graph of the output ontology. The final output ontology is denoted *Ooutput-final*. Table 6 presents the base metrics for the *Obase* and *Ooutput-final* ontologies. It can be noticed in Table 6 the difference between the numbers of classes in the *Obase* and *Ooutput-final* ontologies after a compete execution of the algorithm. The *Ooutput-final* ontology includes 38 classes more than the *Obase* ontology, thereby confirming that this surplus of classes has been added or merged after successful similarity tests.

Table 6. Base metrics of the *Obase* and *Ooutput-final*.

Ontology	# of Classes	# of Logical Axioms	# of Individuals	# of Data Properties	# of Object Properties	TOTAL # of All Axioms
Obase	65	548	17	13	94	1097
Ooutput-final	103	586	17	13	94	1211

It is also shown in Table 6 the differences between the numbers of logical and total axioms. The *Obase* base ontology had 548 logical axioms initially, and after execution, the algorithm achieved a final output ontology with 586 logical axioms. Similarly, the rightmost column of Table 6 indicates a difference of 114 axioms between the total numbers of axioms in the *Obase* base ontology and the *Ooutput-final* final output ontology. These findings attest that the proposed algorithm has successfully tested iteratively and merged many axioms from the candidate ontologies into the final output ontology.

5.4. Analysis of Quality Metrics of Ontologies

A number of graph metrics were used to evaluate and compare the quality of the base, candidate and output ontologies. The graph metrics considered include the Absolute Root Cardinality (ARC), Absolute Leaf Cardinality (ALC), Average Depth (AD), Maximum Depth (MD), Average Breadth (AB) and Maximum Breadth (MB). The ARC, ALC and AD metrics indicate how deep, on average, the taxonomy relations are in the ontology. The MD metric indicates the longest path from the root to the leaf concepts in the ontology. The metrics AB and MB represent the average number of children and the highest number of children per node in the ontology, respectively. The graph metrics of the base, candidate and output ontologies were calculated with the OntoMetrics [36] online ontology quality evaluation platform. The graph metrics for the base ontology and the candidate ontologies are displayed in Table 7, whereas those of the base and output ontologies are depicted in Table 8.

Table 7. Graph metrics for the base and candidate ontologies.

Graph Metric	*Ocandidate-1*	*Ocandidate-2*	*Ocandidate-3*	*Ocandidate-4*	*Obase*
Absolute root cardinality		13	2	12	2
Absolute leaf cardinality		30	46	17	44
Average depth		37	61	21	64
Maximal depth		65	191	32	252
Average breadth		1.710526	3.080645	1.52381	3.452055
Maximal breadth		3	6	3	5
Total number of paths		38	62	21	73
Number of nodes	1859	43	61	40	103
Number of edges	4033	60	60	58	318

Table 8. Graph metrics for the base and new merged ontologies.

Graph Metric	*Obase*	*Ooutput-1*	*Ooutput-2*	*Ooutput-3*	*Ooutput-4*	*Ooutput-final*
ARC	2	2	2	2	2	2
AC	44	46	65	44	44	69
AD	64	70	96	64	64	102
MD	252	284	409	252	252	441
AB	3.452055	3.55	3.787037	3.452055	3.452055	3.834783
MB	5	6	6	5	5	6
Total number of paths	73	80	108	73	73	115
Number of nodes	103	109	135	103	103	141
Number of edges	318	324	350	318	318	356

Table 8 shows that the *Ooutput-final* final output ontology has an ALC of 69, which is higher than the ALC values of the intermediate output ontologies obtained with the execution of the proposed algorithm on each candidate ontology. This indicates that the resulting output ontology is cohesive and has a better understanding compared to the base ontology *Obase* and the intermediate output ontologies *Ooutput-1*, *Ooutput-2*, *Ooutput-3 and Ooutput-4*. Thus, the execution of the proposed algorithm has enriched the base ontology (Code Ontology) with additional relevant data from the *Ocandidate-1* (Software Ontology) and *Ocandidate-2* (Game Ontology) candidate ontologies. The *Ocandidate-2* candidate ontology had the larger number of axioms, concepts and annotations that were merged into the *Ooutput-final final* output ontology, while the *Ocandidate-1* candidate ontology had a few. Another observation in Table 8 is that the depth of the base ontology *Obase* is greater than its breadth. This observation implies that the base ontology includes relevant knowledge within each node (concept).

5.5. Comparison of Base Metrics Obtained with Existing Tools

In order to validate the accuracy of the base metrics of the output ontology generated with the proposed algorithm, we further employed two state-of-the-art tools, namely, OntoMetrics and Protégé. These tools were utilized to generate the base metrics for the output ontologies after the merging process. Table 9 presents these metrics. The comparison of the base metrics of the output ontologies in Tables 2–6, as generated with proposed algorithm, against the base metrics of the output ontologies calculated with OntoMetrics and Protégé (Table 9) revealed a strong correlation between these metrics. This attests to the accuracy and quality of the proposed algorithm.

Table 9. Base metrics of the output ontologies in OntoMetrics and Protégé.

Metrics	OntoMetrics					Protégé				
	Ooutput-1	*Ooutput-2*	*Ooutput-3*	*Ooutput-4*	*Output-final*	*Ooutput-1*	*Ooutput-2*	*Ooutput-3*	*Ooutput-4*	*Output-final*
# of classes	71	97	65	65	103	71	97	65	65	103
# of logical axioms	554	580	548	548	586	554	580	548	548	586
# of individuals	17	17	17	17	17	17	17	17	17	17
# of data properties	13	13	13	13	13	13	13	13	13	13
# of object properties	94	94	94	94	94	94	94	94	94	94
TOTAL # of all axioms	1115	1193	1097	1097	1211	1115	1193	1097	1097	1211

5.6. Consistency Check of Ontologies with Reasoners

To ascertain the consistency of the output ontologies, we used both the Hermit and Pallet reasoners in Protégé. Figures 1 and 2 show snapshots of the consistency checks of

the output ontologies *Ooutput-*$_2$ and *Ooutput-final* with the Hermit and Pellet reasoners, respectively.

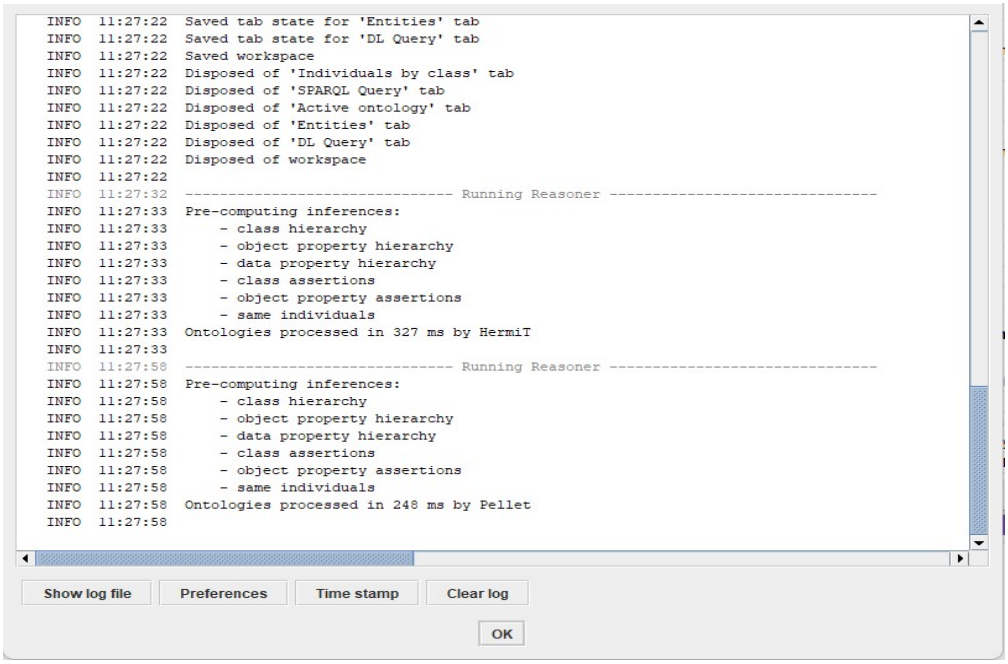

Figure 1. Consistency check of the *Ooutput-2* output ontology with Hermit reasoned with Protégé.

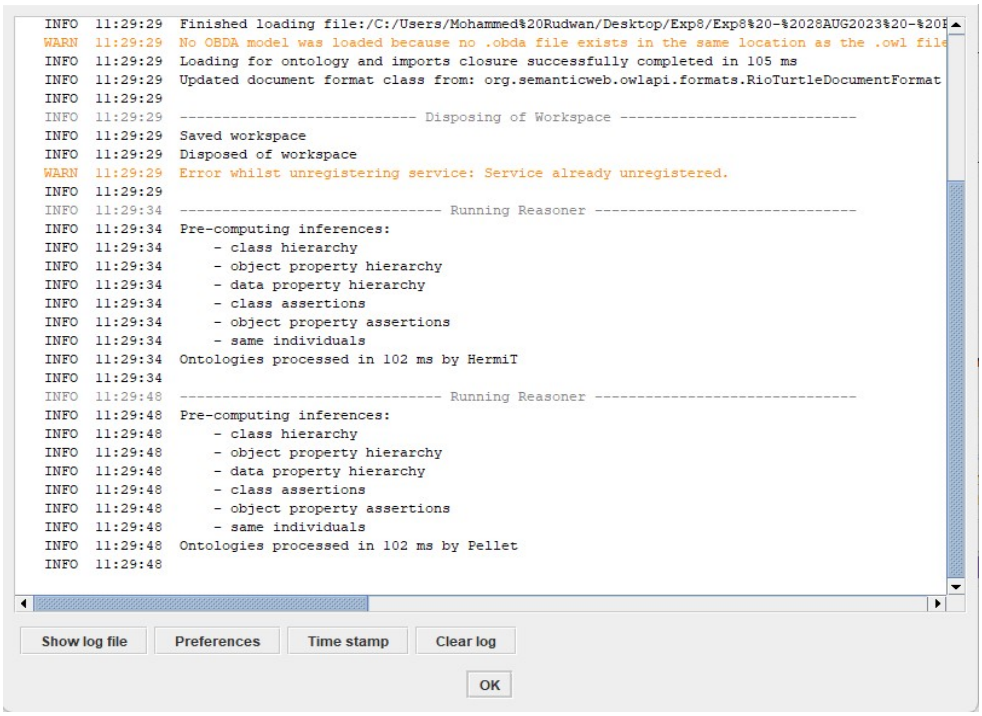

Figure 2. Consistency check of the *Ooutput-final* ontology with the Pellet reasoner within Protégé.

It is shown in the middle and bottom of Figure 1 that all the constituents of the *Ooutput-2* output ontology have been inferred successfully by the Hermit reasoner.

Similarly, the middle and bottom parts of Figure 2 indicate that all the relevant constituents of the *Ooutput-final* final ontology have been successful inferred by the Pellet reasoner. The results showed that there is no single inconsistency detected in the output

ontologies, which indicates that the design and structure of the output ontologies are semantically correct.

5.7. Analysis of Asymptotic Behavior and Computational Efficiency of the Proposed Algorithm

5.7.1. Time Complexity of the Proposed Algorithm

Table 10 shows the time complexity of the set of six algorithms that constitute the proposed algorithm. It is shown in Table 10 that Algorithm 1, which is the main algorithm that encapsulate the other nested algorithms, takes more time to execute. This is because it makes calls to other five functions/algorithms to perform the main three stages of the proposed algorithm.

Table 10. Time complexity of the proposed algorithm.

Algorithm.	Time Complexity
Algorithm 1	$O(n^5)$
Algorithm 2	$O(n^3)$
Algorithm 3	$O(2n)$
Algorithm 4	$O(n-2)$
Algorithm 5	$O(1)$
Algorithm 6	$O(n^2)$

Algorithm 2, which performs the merging of concepts and declarative axioms, is the least efficient among the algorithms, with a time complexity of $O(n^3)$. Algorithm 5, which is used in Algorithm 4, is the best performing algorithm as it runs at a constant time of 1 for all the cases regardless of the input. Algorithm 4, which extracts logical axioms from a given ontology, was found to be the second-best algorithm, with a time complexity of $O(n-2)$, followed by Algorithm 3, which had a time complexity of $O(2n)$. In addition to the fact that Algorithm 1 makes calls to Algorithms 2–6, it includes a main loop (line 11 of Algorithm 1), which in turn includes four nested loops (lines 19, 32, 35 and 36 of Algorithm 1). In the worst-case scenario, the four nested loops of the main loop of Algorithm 1 are executed, which results in a time complexity of $O(n^5)$ for this fragment of Algorithm 1. Therefore, the time complexity of the proposed algorithm (Algorithm 1) is the sum of the time complexities of Algorithms 2–6 and the time complexity of its main loop, which is $O(n^3) + 2O(2n) + O(n-2) + O(n^2) + O(n^5)$. After simplification, the time complexity of the proposed RDF-MOM algorithm is $O(n^5)$.

5.7.2. Analysis of Computational Performance of the Proposed Algorithm

We used an in-code timer to measure the execution time of the proposed algorithm during the merging process. The execution times of the algorithm for merging the base ontology with each of the candidate ontologies were taken separately, as in Table 11.

Table 11. Execution time of the proposed algorithm.

Candidate Ontology	Execution Time (in Seconds)
$Ocandidate\text{-}_1$	6530.91
$Ocandidate\text{-}_2$	4197.87
$Ocandidate\text{-}_3$	23.90
$Ocandidate\text{-}_4$	21.98

Table 11 shows that the proposed algorithm took the least amount of time to merge the *Ocandidate-4* candidate ontology with the *Obase* base ontology. The *Ocandiadte-4* candidate ontology has a smaller size compared to the *Ocandidate-1* candidate ontology that took over an hour to be merged with the base ontology. The *Ocandidate-2* candidate ontology had some similarities with the *Obase* base ontology. In addition, the *Ocandiadte-2* ontology was the second largest in size among the candidate ontologies, with 48 kilobytes. The

big variations between the execution times of the proposed algorithm on the merging of the *Ocandidate-1*, *Ocandidate-3* and *Ocandidate-4* candidate ontologies with the *Obase* base ontology may be attributed to the fact that the algorithm performed some of its worst scenarios while merging the *Ocandiadet-1* candidate ontology with the base ontology.

6. Comparison with Related Works

In this section, we conduct a comparative analysis of our results in relation to other existing studies, shedding light on both the commonalities and differences observed. In Table 13, we provide an overview of how prior research has assessed the proposed methods for ontology merging, the summary of their findings and possible limitations. Additionally, we draw comparisons between our results and those presented in the literature. In Table 1: (1) ✔ indicates that the given criterion is met comprehensively; (2) ✗ signifies the nonfulfillment of the criterion at all; and (3) ≈ denotes an intermediate degree of satisfaction, that is, a partial fulfillment of the criterion.

Table 12. Evaluation of the proposed algorithm's results with related works.

#	Study	Quality Criteria Evaluation				Summary of Findings	Limitation
		Accuracy	Cohesion	Integrity	Running time		
1	[16]	≈	✔	✔	✔	They compared their results in terms of execution time to the HSSM method and concluded that their method outperforms HSSM.	No information about the number of classes in the reference and candidate ontologies. The proposed method compared to only one study. Did not address the accuracy of the output ontologies.
2	[3]	✗	✔	✔	✔		
3	[11]	✗	✔	✔	✗	- They compared their alignment results in terms of similarity scores to others resulting in precision, recall, and f-measure. - Results show that their method (PrOM) outperforms the other 4.	Did not address the accuracy of the output ontology or the running time of the proposed solution.
4	[8]	✔	✔	✔	✗	Their method effectively merge logical	Did not address the running time of the proposed solution
5	[20]	✔	✗	✗	✗	- Their evaluation was against human experts to validate their method's effectiveness. - Their findings support the use of WordNet to obtain semantics of concepts.	Did not address the cohesion and integrity of the output ontologies or the running time of the proposed solution.
6	[1]	✗	✔	✔	✗	They cannot compare their work to another since the OnotMerger they proposed has specific requirements not applicable to others.	Did not address the accuracy of the output ontology or the running time of the proposed solution.
7	[23]	≈	✗	✔	✗	- They used two other metrics to compare their results, namely, COMA++ and FAC.	The comparison was on the base metrics number of classes, property and instances only.

Table 13. Evaluation of the proposed algorithm's results with related works.

#	Study	Quality Criteria Evaluation				Summary of Findings	Limitation
		Accuracy	Cohesion	Integrity	Running time		
8	[18]	✗	✗	✔	✔	- The study preserves all concepts and relationships of the target taxonomy. - Graph metric were used in the evaluation.	Did not address the accuracy and cohesion of the output ontologies.
9	This Study	✔	✔	✔	✔	The proposed algorithm achieved output ontologies that met the 4 quality criteria of accuracy, cohesion, integrity and execution time.	A bid high execution time due to the bigger sizes of candidate ontologies compared to related studies.

Table 13 reveals a prevalent trend among researchers, where a significant portion have not taken into consideration the complete spectrum of quality criteria. For instance, in [20], the only quality criterion examined was accuracy, as the authors sought to validate the reliability of WordNet for semantic concept detection. In both [1,11], the evaluation criteria encompassed cohesion and integrity, with no considerations of the accuracy and execution time performance. Notably, the authors of [1] emphasized that comparing the results achieved by their proposed method, namely, OnotMerger, to those of other works for an overall performance assessment proved challenging. This was attributed to OnotMerger's specific requirements and inputs, which were not aligned with those of other methods.

In contrast, [16] offered a comprehensive evaluation by addressing a wide array of quality criteria, providing compelling evidence that their method outperformed the HSSM approach. However, it is worth noting that their comparative analysis was limited to a single method. Meanwhile, in the case of the ATOM method [18], the authors leveraged graph metrics, particularly by quantifying leaf paths in the knowledge graph, to evaluate their approach. The results highlighted a substantial reduction in the number of leaf paths and total classes, aligning well with the integrity criterion. Furthermore, the study delved into the execution time performance to offer a further perspective on the performance of ATOM.

Compared to the abovementioned related works, our proposed algorithm addressed all four quality criteria outlined in Table 13. Our findings indicated the effectiveness of the proposed algorithm in merging the candidate ontologies with a base ontology. However, the proposed algorithm displayed a higher execution time when aligning large-scale ontologies. Furthermore, as demonstrated in Section 5.7.1, the proposed algorithm may have a time complexity of the order of the fifth degree polynomial in the worst-case scenario. These particular aspects warrant further attention in future research, as we seek to explore and optimize the algorithm for applications involving large datasets, such as big data applications.

7. Conclusions and Future Work

In this study, we aimed to address a broad range of criteria for ontology merging, encompassing concepts (both lexical and semantic), properties, individuals, taxonomy, logical axioms, and annotations. We proposed a new algorithm that performs ontology merging in three stages, namely, concept and annotations, logical axioms, and individuals. This algorithm utilizes RDF graphs to iteratively merge the candidate ontology with the base ontology while preserving the structural integrity of the base ontology. In the first stage, the merging of concepts and annotations is achieved through the utilization of a Machine Learning-based framework called SM-DTR, WordNet, and BERT. Subsequently,

the updated RDF graph serves as the input for the second stage, where logical axioms from the base and candidate ontologies are aligned and merged using the Word2Vec model and Jaro–Winkler algorithm. The resulting RDF graph then proceeds to the final stage, where individual merging occurs, facilitated by the Jaccard fuzzy string-matching algorithm. To assess the algorithm's performance, we conducted asymptotic analysis and computational evaluations, along with the extraction of the base and graph metrics from the resulting output ontologies to evaluate their quality. While the findings revealed that the proposed algorithm is time-consuming according to the time complexity analysis and running time results, the quality of the resultant ontologies significantly improved. These resulting output ontologies met the established ontology-merging quality criteria, including integrity, cohesion, and accuracy. Furthermore, we compared the results achieved by the proposed algorithm with previous endeavors in the literature and found that our approach comprehensively addresses the quality criteria, unlike many existing studies. Future efforts will focus on enhancing the proposed algorithm by offering configurable settings that allow users to select alternative algorithms and techniques for each merging stage, tailoring the algorithm to specific user needs. Additionally, we will explore improvements in computational performance to support big data applications.

Author Contributions: Conceptualization, M.S.M.R. and J.V.F.-D.; methodology, M.S.M.R. and J.V.F.-D.; software, M.S.M.R.; validation M.S.M.R. and J.V.F.-D.; formal analysis, M.S.M.R. and J.V.F.-D.; investigation, M.S.M.R. and J.V.F.-D.; resources, M.S.M.R.; data curation, M.S.M.R. and J.V.F.-D.; writing—original draft preparation, M.S.M.R.; writing—review and editing, M.S.M.R. and J.V.F.-D.; visualization, M.S.M.R. and J.V.F.-D.; supervision, J.V.F.-D.; project administration, M.S.M.R. and J.V.F.-D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The ontologies that constitute the dataset used in this study can be found at the following links: https://zenodo.org/records/577939; https://www.ebi.ac.uk/ols/ontologies/swo; http://vocab.linkeddata.es/vgo/; http://iot.ee.surrey.ac.uk/iot-crawler/ontology/iot-stream/.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Geleta, D.; Nikolov, A.; Odonoghue, M.; Rozemberczki, B.; Gogleva, A.; Tamma, V.; Payne, T.R. OntoMerger: An ontology integration library for deduplicating and connecting knowledge graph nodes. *arXiv* **2022**, arXiv:2206.02238.
2. Farghaly, K.; Soman, R.K.; Collinge, W.; Mosleh, M.H.; Manu, P.; Cheung, C.M. Construction safety ontology development and alignment with industry foundation classes (IFC). *J. Inf. Technol. Constr.* **2022**, *27*, 94–108. [CrossRef]
3. Priya, M.; Ch, A.K. A novel method for merging academic social network ontologies using formal concept analysis and hybrid semantic similarity measure. *Libr. Hi Tech* **2020**, *38*, 399–419. [CrossRef]
4. Milani, M.K.; Hashemi, M.R. Extended grounded theory: A methodology to combine multiple disciplines. *Inf. Syst. e-Bus. Manag.* **2020**, *18*, 89–120. [CrossRef]
5. Jachimczyk, B.; Tkaczyk, R.; Piotrowski, T.; Johansson, S.; Kulesza, W. IoT-based dairy supply chain-an ontological approach. *Elektron. Ir Elektrotechnika* **2021**, *27*, 71–83. [CrossRef]
6. Reed, L.; Harrison, V.; Oraby, S.; Hakkani-Tur, D.; Walker, M. Learning from mistakes: Combining ontologies via self-training for dialogue generation. *arXiv* **2020**, arXiv:2010.00150.
7. Mao, Q.; Li, X.; Peng, H.; Li, J.; He, D.; Guo, S.; He, M.; Wang, L. Event prediction based on evolutionary event ontology knowledge. *Future Gener. Comput. Syst.* **2021**, *115*, 76–89. [CrossRef]
8. Gueddes, A.; Mahjoub, M.A. *A Jena API for Combining Ontologies and Bayesian Object-Oriented Networks*; IEEE: Piscataway, NJ, USA, 2022; pp. 355–360.
9. Shi, J.; Pan, Z.; Jiang, L.; Zhai, X. An ontology-based methodology to establish city information model of digital twin city by merging BIM, GIS and IoT. *Adv. Eng. Inform.* **2023**, *27*, 102114. [CrossRef]
10. Zhang, X.-Z.; Yu, B.-H.; Liu, C. SSN_SEM: Design and application of a fusion ontology in the field of medical equipment. *Procedia Comput. Sci.* **2021**, *183*, 677–682. [CrossRef]
11. Ocker, F.; Vogel-Heuser, B.; Paredis, C.J.J. A framework for merging ontologies in the context of smart factories. *Comput. Ind.* **2022**, *135*, 103571. [CrossRef]
12. Babalou, S.; König-Ries, B. GMRs: Reconciliation of Generic Merge Requirements in Ontology Integration. In *SEMANTiCS Posters&Demos*. 2019. Available online: https://ceur-ws.org/Vol-2451/paper-04.pdf (accessed on 20 February 2024).

13. Fonou-Dombeu, J.V.; Viriri, S. *OntoMetrics Evaluation of Quality of e-Government Ontologies*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 189–203.
14. Osman, I.; Yahia, S.B.; Diallo, G. Ontology integration: Approaches and challenging issues. *Inf. Fusion* **2021**, *71*, 38–63. [CrossRef]
15. Tartir, S.; Arpinar, I.B.; Sheth, A.P. Ontological evaluation and validation. In *Theory and Applications of Ontology: Computer Applications*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 115–130.
16. Priya, M.; Aswani Kumar, C. An approach to merge domain ontologies using granular computing. *Granul. Comput.* **2021**, *6*, 69–94. [CrossRef]
17. Raunich, S.; Rahm, E. Target-driven merging of taxonomies with ATOM. *Inf. Syst.* **2014**, *42*, 1–14. [CrossRef]
18. Madani, K.; Russo, C.; Rinaldi, A.M. *Merging Large Ontologies Using Bigdata Graphdb*; IEEE: Piscataway, NJ, USA, 2019; pp. 2383–2392.
19. Robin, C.R.R.; Uma, G.V. A novel algorithm for fully automated ontology merging using hybrid strategy. *Eur. J. Sci. Res.* **2010**, *47*, 74–81.
20. Amrouch, S.; Mostefai, S. *Syntactico-Semantic Algorithm for Automatic Ontology Merging*; IEEE: Piscataway, NJ, USA, 2012; pp. 1–5.
21. Hnatkowska, B.; Kozierkiewicz, A.; Pietranik, M. Semi-automatic definition of attribute semantics for the purpose of ontology integration. *IEEE Access* **2020**, *8*, 107272–107284. [CrossRef]
22. Mountasser, I.; Ouhbi, B.; Hdioud, F.; Frikh, B. Semantic-based Big Data integration framework using scalable distributed ontology matching strategy. *Distrib. Parallel Databases* **2021**, *39*, 891–937. [CrossRef]
23. Maiz, N.; Fahad, M.; Boussaid, O.; Bentayeb, F. Automatic Ontology Merging by Hierarchical Clustering and Inference Mechanisms. In Proceedings of the I-KNOW 2010, Graz, Austria, 1–3 September 2010; pp. 1–3.
24. Rudwan, M.S.M.; Fonou-Dombeu, J.V. *Machine Learning Selection of Candidate Ontologies for Automatic Extraction of Context Words and Axioms from Ontology Corpus*; Springer: Berlin/Heidelberg, Germany; pp. 282–294.
25. Gruber, M.; Eichstädt, S.; Neumann, J.; Paschke, A. Semantic information in sensor networks: How to combine existing ontologies, vocabularies and data schemes to fit a metrology use case. In Proceedings of the 2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT, Roma, Italy, 3–5 June 2020; pp. 469–473.
26. Cheatham, M.; Hitzler, P. String similarity metrics for ontology alignment. In Proceedings of the Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, Australia, 21–25 October 2013; pp. 294–309.
27. Rudwan, M.S.M.; Fonou-Dombeu, J.V. Hybridizing Fuzzy String Matching and Machine Learning for Improved Ontology Alignment. *Future Internet* **2023**, *15*, 229. [CrossRef]
28. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
29. Miller, G.A. WordNet: A lexical database for English. *Commun. ACM* **1995**, *38*, 39–41. [CrossRef]
30. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
31. Atzeni, M.; Atzori, M. Object-Oriented Programming Languages and Source Code. 2017. Available online: https://zenodo.org/records/577939 (accessed on 20 February 2024).
32. Institute, E.B. 2022. Software Ontology. Available online: https://www.ebi.ac.uk/ols/ontologies/swo (accessed on 20 February 2024).
33. The Video Game Ontology Version 3. 2016. Available online: http://vocab.linkeddata.es/vgo/ (accessed on 20 February 2024).
34. Surrey, U.O. IoT-Stream: A Lightweight Ontology for IoT Data Streams. 2019. Available online: http://iot.ee.surrey.ac.uk/iot-crawler/ontology/iot-stream/ (accessed on 20 February 2024).
35. Lohmann, S.; Negru, S.; Haag, F.; Ertl, T. Visualizing ontologies with VOWL. *Semant. Web* **2016**, *7*, 399–419. [CrossRef]
36. Lantow, B. OntoMetrics: Putting Metrics into Use for Ontology Evaluation. In Proceedings of the 8th International Joint Conference, IC3K 2016, Porto, Portugal, 9–11 November 2016; pp. 186–191.
37. Musen, M.A. The protégé project: A look back and a look forward. *AI Matters* **2015**, *1*, 4–12. [CrossRef]