Adapting semantic similarity methods for case-based reasoning in the Cloud.

NKISI-ORJI, I., PALIHAWADANA, C., WIRATUNGA, N., CORSAR, D. and WIJEKOON, A.

2022

Adapting Semantic Similarity Methods for Case-Based Reasoning in the Cloud *

Ikechukwu Nkisi-Orji[0000−0001−9734−9978], Chamath Palihawadana[0000−0002−9594−8683], Nirmalie Wiratunga[0000−0003−4040−2496], David Corsar[0000−0001−7059−4594], and Anjana Wijekoon[0000−0003−3848−3100]

School of Computing, Robert Gordon University, Aberdeen, UK
{i.nkisi-orji,c.palihawadana,n.wiratunga,d.corsar1,a.wijekoon1}@rgu.ac.uk

Abstract. CLOOD is a cloud-based CBR framework based on a microservices architecture which facilitates the design and deployment of case-based reasoning applications of various sizes. This paper presents advances to the similarity module of CLOOD through the inclusion of enhanced similarity metrics such as word embedding and ontology-based similarity measures. Being cloud-based, costs can significantly increase if the use of resources such as storage and data transfer are not optimised. Accordingly, we discuss and compare alternative design decisions and provide justification for each chosen approach for CLOOD.

Keywords: CBR architectures and frameworks, Cloud microservices, Semantic similarity, Ontologies

1 Introduction

Modern computing has evolved rapidly with the adoption of cloud computing across sectors such as healthcare, finance and travel. As organisations transition to the latest technology stacks and update the underpinning infrastructure, keeping existing development tools and frameworks up to date can pose significant challenges. Case-based reasoning (CBR) is an example of one such application area where once popular frameworks have become unusable with new, state-of-the-art technologies. Recently CLOOD [13] was introduced with the aim of addressing this challenge by adopting a cloud-first approach based on a microservices oriented architecture.

CLOOD has demonstrated more robust performance and scalability than other CBR systems like jCOLIBRI[4] and has been successfully integrated into industry projects, enabling the application of aspects of the CBR cycle in large scale systems. The practical demands of applying CBR in real-world applications often leads to the requirement to extend the built-in local similarity functions. One such project is *iSee*[1] which aims to enhance the explainability of machine

* This research is funded by the iSee project (https://isee4xai.com) which received funding from EPSRC under the grant number EP/V061755/1. iSee is part of the CHIST-ERA pathfinder programme for European coordinated research on future and emerging information and communication technologies.

[1] https://isee4xai.com/

learning models. *iSee* is developing a CBR system that uses the experiences of multiple users who have received explanations for the outputs of machine learning models, as the basis for new explanation strategies (combinations of explainability techniques), generated using CBR methods. The new strategy can then be executed to provide new users with explanations for model output(s). The *iSee* project is developing iSeeOnto[2] to describe AI Models, explanations, explainability techniques, explanation strategies, and user experiences. We argue that the similarity and adaptation containers of the *iSee* CBR system would benefit from using advanced semantic similarity methods, such as ontology-based and vector-based approaches to improve the quality of the generated explanation strategies.

CLOOD models its casebase using an open-source search engine (OpenSearch[3]) which leverages the efficient Lucene index for case retrieval. While the similarity module of OpenSearch (and similar search engines) supports several similarity metrics such as exact and fuzzy matching techniques out-of-the-box, it lacks several similarity metrics that are useful for case-based reasoning applications such as *iSee*. Extending the similarity module to include new metrics requires the use of custom similarity scripts that accomplish the retrieval needs within the microservices architecture and without a significant overhead in resource use.

A key challenge when extending the capabilities of a framework like CLOOD is to retain the core design principles: cloud-first, microservices-oriented, and serverless (on-demand workloads) application model. Furthermore, a framework will be sustainable and adopted by the community only if it maintains the operational overheads (e.g. compute time and resource cost) as new features are added. In this work, both of these challenges are addressed while enhancing the functionality of the CLOOD CBR framework with support for semantic similarity metrics for local similarity which includes:

- similarity table, word embedding based similarity, and ontology-based similarity measures,
- architectural considerations to retain the microservices nature of the platform,
- minimising the retrieval overhead due to the introduction of the semantic similarity measures,
- reviewing the potential impact of extending CLOOD as a platform for rapid integration of CBR.

The remainder of this paper is structured as follows; in Section 2 we discuss the related work in semantic relatedness measures, ontologies in CBR and CLOOD; Section 3 presents the introduced semantic similarity metrics; Section 4 discusses the integration of the metrics into the existing microservices architecture of CLOOD; Section 5 experimentally evaluates the impact of adopting alternative approach on the most resource-intensive metrics; Section 6 concludes the paper with a review of contributions and outlines future directions of CLOOD.

[2] https://github.com/isee4xai/iSeeOnto
[3] `https://opensearch.org/`

2 Related Work

2.1 Clood CBR

Introduced in [13], CLOOD is the first cloud-based generic CBR framework developed for scalability using a microservices-oriented design. Prior to CLOOD, CBR frameworks were based on monolithic architectures which was restrictive for applications that require high scalability. CLOOD opened up a new avenue in the CBR landscape by enabling CBR applications to reach higher levels of scale. An empirical study showed that CLOOD can scale extensively without compromising performance (e.g. retrieval from a casebase of half a million cases was over 3,700 times faster than jCOLIBRI). Further, limited integration support (e.g. APIs) in the previous CBR frameworks has been an obstacle for adopting CBR for some real-world applications. The extensible and open source nature of CLOOD has facilitated its improvement and enhancement over time. Implementation using the microservices paradigm makes CLOOD more sustainable considering the rapid adoption of cloud computing and serverless computing, particularly in enterprise settings [19]. A complete CLOOD based implementation consists of the following:

- **Casebase** contains the cases of a CBR application and is implemented using a full-text search engine (e.g. Elasticsearch, OpenSearch).
- **CBR Functions** consists of the CBR cycle operations (retrieve, reuse, revise, retain), which were implemented using Serverless functions.
- **Similarity Functions** provides the mechanism to match and rank cases during case retrieval. The microservices based architecture in CLOOD is advantageous for extending similarity functions.
- **External Access** consists of a set of HTTP API endpoints that trigger the CBR functions. This external interface is implemented using cloud-native microservices that is available on most cloud service providers (e.g. AWS API gateway, Google API Gateway).
- **Data sources** are the persistence tools where applications store their data (e.g. MySQL, MongoDB) and these data sources can be synchronised with the Casebase through the external access APIs.
- **Client applications** are front facing components of any system such as the CLOOD dashboard. Client applications on any platform can use the external APIs to achieve CBR integration.

2.2 Ontologies in CBR

Ontologies have long been used in various aspects of CBR such as the vocabulary for describing cases [1, 12, 16], case structure and indexing [14], semantic knowledge for similarity measurements [1, 6, 12, 14, 16], and domain knowledge for case adaptation [16]. Our focus is the use of ontologies to determine the semantic similarity during case retrieval. Ontology-based semantic similarity approaches consider factors such as the taxonomic relations between concepts and

the degree of shared properties. The Ontology-based similarity approaches differ according to the type of information used to determine similarity (or relatedness) with path based, information content (IC) based and feature based as the main categories of alternative approaches [9]. Path based approaches rely on relative distances between the concepts of an ontology to determine similarity [15, 21]. The first path based methods solely relied on the shortest distance between concepts. Subsequent methods added extensions such as the use of most specific common subsumer (MSCS) information and depth scaling. MSCS is the most distant node from the root that subsumes the concepts of a comparison. Typically, concepts become increasingly specific as an ontology is traversed from the root to leaf nodes. Depth-scaling approaches consider concepts in close proximity to be more closely related when they are nearer the leave nodes than when they are nearer the root. IC approaches integrate a measure of information content in their similarity models [10, 17]. Some IC approaches rely on the occurrence information of concepts in an external corpus to determine their information content and this introduces the challenge of correctly annotating the corpus. Feature based approaches determine similarity by comparing the features of concepts [18]. Similarity is determined by offsetting the degree of common features by the distinct features. The symmetric property is one of the differentiating features of different algorithms for ontology-based similarity. With the symmetric property, the similarity between concepts remains the same irrespective of the direction of comparison (i.e. $sim(x, y) = sim(y, x)$). Ontology-based similarity or relatedness measures correlate positively with human judgements to a good extent [7].

2.3 Retrieval with Word Embedding

A word embedding is a distributed representation for text that allows words with similar meanings to have similar embeddings. Most of the recent word embedding methods use neural networks to generate embedding vectors (dense vector of real numbers) that encode the context-induced meaning of words [11]. Context is determined by a corpus from which the word embedding is generated. Similarity is computed by comparing the vector representation of terms with the expectation that words with similar meanings will have similar embeddings. There are different types of learning techniques for creating word embedding models (e.g. Word2Vec, GloVe, BERT) [8] and several pre-trained models are available for reused. The embedding vectors can be used to determine local similarity measures during case retrieval as discussed in [2].

The use of vector-based similarity is not limited to textual CBR as it can apply to other media (e.g. image and audio). Also, there are ontology-based similarity approaches that use word embedding techniques to embed knowledge graphs into a dense vector space [9].

2.4 Serverless Function Benefits and Limitations

Serverless concepts provide many benefits such as the ability to scale systems according to workloads, flexibility of development, isolation of services, minimum maintenance overheads and major cost reductions. However, there are a few drawbacks which can pose implementation challenges such as provider-defined limits on compute resources (e.g. execution timeout in AWS Lambda, a widely used Function-as-a-Service) [22, 20]. Serverless functions are used to compute fine-grained tasks which require very little computation unlike a large monolith application. Based on the study in [5], scalability is the main goal when using the microservices architecture while performance and response time are the chief optimisation concerns.

Clood depends on serverless functions for similarity calculations and other CBR functions, which only require small amounts of computation resources. Also, the casebase is not loaded into memory in Clood for scalability and performance reasons. However, operations such as pre-loading a large database from a CSV file, generating vector representations of strings, or computing similarity between nodes of a logically complex ontology will require additional computational resources, which can increase costs and affect the overall system performance. In this work we explore these limitations and propose options to overcome the challenges of using rich knowledge sources (word embedding and ontology) for similarity measures in Clood without imposing a significant negative impact on scalability, performance and response time.

3 Semantic Similarity Metrics in a Microservices Architecture

Several design challenges have to be overcome in order to implement enhanced similarity measures in the microservices architecture which Clood uses. In this section we present how the current limitations of cloud-based microservice architectures can be overcome to enable the inclusion of semantically rich knowledge resources for similarly measures. Specifically, we consider similarity metrics using similarity tables, word embeddings and ontologies. By leveraging the indexing capabilities of a NoSQL database equipped with a search engine, the current and future similarity methods can be incorporated into Clood without sacrificing overall CBR performance.

3.1 Clood Similarity Functions Overview

Table 1 is an overview of the local similarity metric functions that are available on Clood. This paper focuses on the similarity metrics in bold font face. The similarity metrics that are not marked as "Core" are implemented as separate microservices that are used by the base retrieval system through API calls during case retrieval. The decision to implement these metrics as separate services due to their resource needs is one of the measures used to ensure efficiency in performance.

Table 1. CLOOD's Local Similarity Metrics - the enhanced metrics presented in this paper are highlighted in bold text.

Data type	Similarity metric	Description	Core
All	Equal	Similarity based on exact match (used as a filter)	✓
String	EqualIgnoreCase	Case-insensitive string matching	✓
	BM25	TF-IDF-like similarity based on Okapi BM25 ranking function	✓
	Semantic USE	**Similarity measure based on word embedding vector representations**	–
Numeric	Interval	Similarity between numbers in an interval	✓
	INRECA	Similarities using INRECA More is Better and Less is Better algorithms	✓
	McSherry	Similarities using McSherry More is Better and Less is Better algorithms	✓
	Nearest Number	Similarity between numbers using a linear decay function	✓
Categorical	EnumDistance	Similarity of values based on relative positions in an enumeration	✓
	Table	**User-defined similarity between entries of a finite set of domain values**	✓
Date	Nearest Date	Similarity between dates using a decay function	✓
Location	Nearest Location	Similarity based on separation distance of geo-coordinates using a decay function	✓
Ontology	**Path-based**	**Similarity using Wu & Palmer path-based algorithm**	–
	Feature-based	**Similarity using Sanchez et al. feature-based algorithm**	–

3.2 Similarity Table

A similarity table is a cartesian square of similarity measures of a finite set domain values/entities that specifies how any pair of values are related. A similarity table can capture the knowledge of a domain expert by recording their assessment of the similarity of entity pairs. Practical considerations mean this approach is best suited when the number of values is low. The complexity for specifying the similarity knowledge for the similarity table is $O(n^2)$ which can make managing the table to become tedious and time-consuming for domain experts as the number of possible values n, increases. The number of new similarity measures to be specified increases by $2n + 1$ whenever n increases by 1 for asymmetric similarity. When relying on domain experts to explicitly create similarity tables is unfeasible, word embeddings and ontology-based similarity measures can be used to reduce the burden of acquiring the semantic similarity knowledge for case retrieval.

Since user-defined similarity tables are not expected to grow too large, each similarity table is persisted as part of the similarity knowledge for case retrieval. The similarity table forms a lookup table for the local similarity measures between query and case values at retrieval time.

3.3 Word Embedding Based Similarity

A word embedding based similarity method can be added to the CBR system as a loosely-coupled service. Persisting the word embedding model, generating embedding vectors for terms and comparing the vector representations are the main resource considerations for using a similarity measure based on Word Embedding. By using a separate service, we ensure that the word embedding component is easier to manage and able scale up or down according to the model's requirements without affecting the other components. Also, adding the component as a separate service offers the flexibility to exclude it from an instantiation of the CBR framework when it is not needed leading to further reduction in resource use and lower-cost application when deployed on cloud infrastructure.

The processes that run on serverless functions are expected to be ephemeral and free up resources as the execution of functions end. When using word embedding model, most of the time is spent to load up the model and reloading the model each time we want to compute similarity measures can cause significant delays. In order to minimise delays during case retrieval, we generate any required vector representations when cases are being added during the pre-cycle stage or during a retain operation. The resulting dense vector representations are persisted as part of the retrieval knowledge in the casebase. While pre-computing vector representations slightly increases the time it takes to add cases to the casebase, it significantly reduces the time it takes to retrieve cases. At retrieval time, we only determine the vector representations of the applicable query values and compute word embedding based local similarity measures by taking the cosine similarity between the query vectors and the pre-loaded vector representations of cases.

3.4 Ontology-based Similarity Measure

An ontology-based similarity method can be integrated into the microservices architecture as a separate service that is accessed by the base system through API calls. Similar to the word embedding similarity component, this separation increases flexibility and allows the service is manage its resource needs independent of the base system. Accordingly, there were two main challenges to overcome when implementing ontology-based similarity: how to maintain the ontology-based similarity independent in a scalable manner; and how computationally intensive operations like large tree traversal can be implemented using cloud functions.

On the first challenge, a key decision was made to change the system architecture to support computing similarity measures outside the base retrieval

system. Prior to this challenge, all the similarity function scripts were maintained inside the core retrieval functions, but for enhanced similarity methods like ontology-based similarity, support for the use of local similarity measures that are returned through API responses was needed. This requirement is because, there was no practical way of directly apply ontology-based similarity computation to case retrieval without having to load the cases into memory and this will have significant negative impact on the system's scalability. Instead, the ontology-based similarity component computes and returns all the similarity measures that are relevant for a query as a similarity table that is looked up for local similarity measures at retrieval time.

Given an ontology, θ with concepts $c_i, \ldots, c_n$, the similarity table for all the possible similarity measures of the n concepts require n^2 comparisons. However, when the query concept $c_q \in \theta$ is known, we only need to a row of the similarity table that compares c_q with the other concepts of θ for case retrieval (i.e. n comparisons). In addition to reducing computation cost, using a row of the similarity table for each retrieval instance reduces the data transfer between the ontology-based similarity service and the base retrieval system.

In considering the computational cost that is associated with using ontology-based similarity methods and knowing that we only need a subset of ontology concept comparisons when retrieving for a query, we considered three options:

1. **Pre-loaded**: Pre-load determines all the similarity measures that can be determined by the ontology and persists the measures in a searchable index (flattened). We expect this method to have high a computation cost for large ontologies. The pre-loading of similarity measures takes place only once after the ontology added. Also, any updates to the ontology will require a complete refresh of the index to determine new similarity measures.
2. **Cached**: Similarity knowledge for retrieving for any newly seen concept (i.e. similarity between query concept and all the concepts of the ontology) is computed and cached in a searchable index. Whenever a request is made with a query concept, new ontology-based similarity measures are computed only if it is not stored previously in the index. Otherwise, it performs a simple lookup of previously cached similarity measures. Any updates to the ontology will require clearing the cached similarity measures.
3. **Non-Cached**: This approach will not persist any similarity measures so that similarity computation occurs on each request. This method will not use up storage space for an index.

An enhancement that can improve efficiency and reduce retrieval times is the ability to limit similarity computations to a sub-graph of the ontology. For example, a travel ontology can specify entities for both accommodation and activity amongst others. If we are only interested in the activity entities for case description, the ability to limit similarity computations to the activity sub-graph of the ontology can result in significant savings with respect to computation resources and retrieval times.

Note that the ontology-based similarity measure discussed here is applicable when the case attribute values map to classes (or concepts) instead of instances

of classes. However, the ontology-based similarity component can be extended to support the computation of similarity measures for instances of classes.

4 Implementation of Semantic Similarity Measures on Clood Framework

This section discusses how the semantic similarity measures are implemented in CLOOD taking the design consideration in Section 3 into account. Figure 1 presents the CLOOD system architecture with its microservices. Updated com-

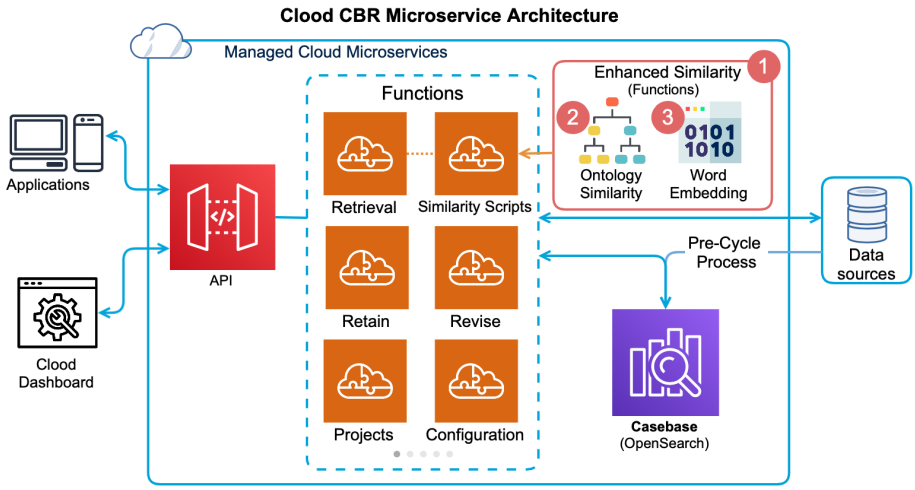

Fig. 1. Updated CLOOD CBR Architecture diagram

ponents of the diagram are numbered as from 1-3. Component (1) represents the proposed isolation of enhanced similarity functions, this component represents all the extended functions as microservices. Component (2) and (3) represents the proposed Ontology-based similarity measure (Section 3.4) and Word Embedding similarity measure (Section 3.3) respectively. The discussion in this section includes how the similarity measures are used through the CLOOD dashboard. As indicated in Figure 1, the user interfaces can be implemented differently with API access to the CLOOD functions. Description of the API end-points are available on the CLOOD repository[4].

Building a similarity table on CLOOD is facilitated by an interactive user interface that enables a user to enumerate the possible values that attributes and the pairwise similarity measures of values as shown in Figure 2. The configuration interface to create a similarity table is only available for attributes with

[4] `https://github.com/RGU-Computing/clood`

table similarity metric type. There is option to make the similarity table symmetric or asymmetric. The user is required to provide only half of the similarity measures when using the symmetric option. The similarity table is persisted as

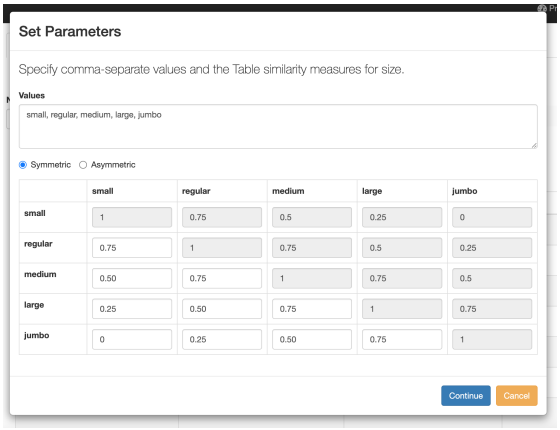

Fig. 2. Specifying a similarity table on CLOOD.

a parameter of the case definition/configuration for lookup during case retrieval. The case definition is where the similarity type, reuse strategy and feature importance/weight of each case attribute is held for computing and aggregating local similarity measures, and for composing a recommended solution.

4.1 Word Embedding Similarity on Clood

We implemented semantic similarity on CLOOD using a pretrained Universal Sentence Encoders (USE) for word embedding[5]. USE is a general-purpose context-aware encoding model for word representations based on the transformer architecture [3]. Additional word embedding models can be included and the USE model can be replaced by a different type of word embedding model. The vector representations are stored as dense vector types in the casebase and the built-in functionality of OpenSearch is used to determine the cosine similarity between vector representations for local similarity measures during case retrieval.

Although similarity is based on vector representations when using word embedding, presenting the dense vectors of real numbers to users will not be very useful. Accordingly, appropriate conversions are made in the layer between the user interface and the casebase. In order to support this conversion, the values of attributes with word embedding vector based similarity are persisted as object (non-primitive) data type in the casebase consisting of both the human-readable attribute values and their corresponding vector representations. The user only

[5] `https://npmjs.com/package/@tensorflow-models/universal-sentence-encoder`

see the human-readable values while the retrieval function uses the vector representations to match and rank cases in the casebase. As a result, no modifications were made to the user interfaces to support the use of word embedding based similarity measures. The user only has to indicate that an attribute will use a word embedding similarity metric function during the attribute configuration for a casebase.

4.2 Ontology-based Similarity on Clood

We implemented the service for ontology-based semantic similarity measures on Clood with the options for Pre-loaded, Cached and Non-Cached methods as discussed in Section 3.4. If the user specifies that an attribute will use an ontology-based similarity approach for local similarity, a configuration option to specify the ontology detail becomes available on the user interface. The configuration interface allows the user to specify the URI locations of the ontology files, each file's format and an optional name for the ontology source. The support for including multiple sources/files is useful for instances where an ontology is stored in multiple files of overlapping concepts (e.g. the modular structure of iSeeOnto). There is option to specify a root concept which is useful when the intention is to use a sub-graph of the ontology for computing similarity measures. When a root is not specified, it is inferred as the uppermost reachable node from the concepts of the ontology. Also, there is option to specify a relation type for determining taxonomic structure if it is different from "rdfs:subClassOf" relation.

We demonstrated support for ontology-based semantic similarity measures by integrating path-based and feature-based similarity measures on Clood. Path-based approach uses Wu and Palmer algorithm [21] as shown in Equation 1

$$sim(c_i, c_j)_{wup} = \frac{2 * depth(mscs(c_i, c_j))}{depth(c_i) + depth(c_j)} \tag{1}$$

where $mscs(c_i, c_j)$ is the most specific common subsumer of concepts c_i and c_j. The depth of concept c_i (i.e. $depth(c_i)$) is the number of edges between c_i and the root node.

Feature-based approach is based on Sanchez et al. normalised dissimilarity [18] as shown in Equation 2

$$sim(c_i, c_j)_{san} = 1 - \log_2(1 + \frac{|\phi(c_i) \setminus |\phi(c_j)| + |\phi(c_j) \setminus |\phi(c_i)|}{|\phi(c_i) \setminus |\phi(c_j)| + |\phi(c_j) \setminus |\phi(c_i)| + |\phi(c_i) \cap |\phi(c_j)|}) \tag{2}$$

where $\phi(c)$ is the taxonomic ancestors of concept c.

5 Evaluation of Resource Impact

In considering the semantic similarity measures we have discussed, design decision based difference in performance is expected to be most noticeable for ontology-based similarity measures. Accordingly, we compare the three implementation options (i.e. pre-loaded, cached or non-cached) to highlight their strengths and weaknesses.

5.1 Experiment Setup

We use the Pizza Price Prediction dataset[6] that describes pizza using attributes for company, price, topping, size, etc. for the case description. We use the pizza ontology[7] for the ontology-based similarity of different types of pizza toppings during case retrieval. First, we anchor the dataset attribute's pizza topping values to the concepts of the pizza ontology. To achieve this, we used an edit distance similarity measure (based on Levenshtein) to map pizza toppings of the dataset to the most suitable concepts of the pizza ontology. The topping attribute was set up to CLOOD's path-based similarity to determine local similarity measures during retrieval. Then, we randomly selected 500 toppings with replacement from the pizza ontology to compose queries for case retrieval. Random selection with replacement ensures that queries can repeat multiple times which is expected in real-life applications. Several toppings repeat multiple times in the query selection because the ontology specifies 48 unique pizza toppings. Figure 4 shows the proportion of the query values that are unique as the query selection increases. At 500 query selection, all the pizza toppings had been seen more once as indicated by the line that shows the proportion of unique queries go to 0.

5.2 Result and Discussion

We performed case retrieval and noted the retrieval times as the number of queries increase as shown in Figure 3. Retrieval times were measured at different intervals and the average time for 5 runs of each set of queries was reported.

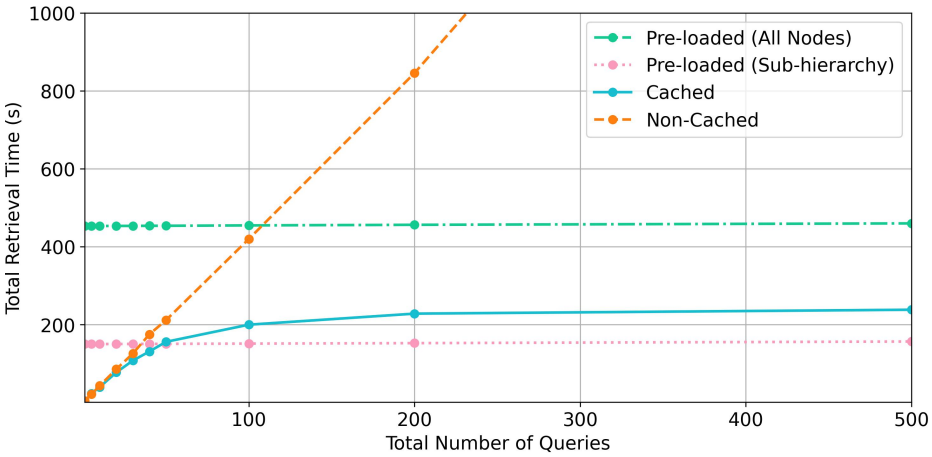

Fig. 3. Times for case retrieval for different methods of implementing ontology-based similarity.

[6] https://www.kaggle.com/datasets/knightbearr/pizza-price-prediction
[7] `https://protege.stanford.edu/ontologies/pizza/pizza.owl`

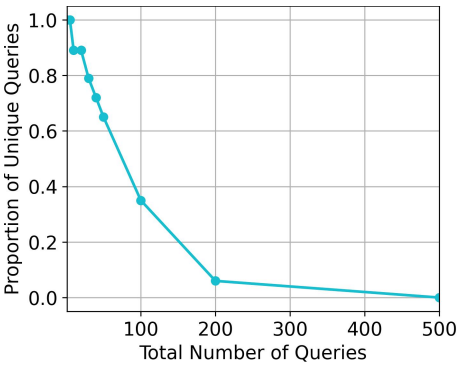

Fig. 4. Proportion of unique queries with total number of queries

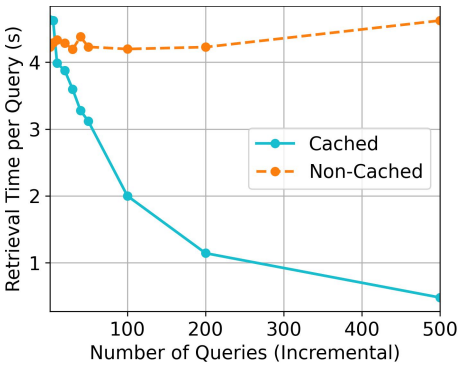

Fig. 5. Comparison between Cache and Non-Cache methods

In Pre-loaded, most of the time is spent to pre-compute all the pairwise similarity measures between the ontology concepts. In the experiment, we also compared the using the entire ontology graph (i.e. Pre-loaded (All Nodes)) with the option that pre-loads only a sub-hierarchy of the ontology (i.e. Pre-loaded (Sub-hierarchy)). For the sub-hierarchy, we used a portion of the ontology for pizza toppings only by specifying the most general concept of pizza topping as the root for similarity measures. Specifying the root restricted computations to a smaller relevant portion of the ontology which reduced the total retrieval times. Given that there are 48 pizza toppings in the ontology, the pairwise similarity computation determines 2,304 (48 x 48) similarity measures when using the sub-hierarchy option. The entire ontology describes 99 concepts (classes) with a possible 9,801 pairwise similarity measures most of which are outside the pizza topping sub-domain. We added the pre-loading time to the retrieval time of the first query. The time spent in executing additional queries mostly consists of the lookup time from the index of similarity measures. In the Cached option, the total retrieval time is initially similar to Non-Cache. However, it becomes almost horizontal as queries begin to recur. At some point, all the concepts become available in the cache so that it becomes a lookup of similarity values like the pre-loaded option. The Non-Cached option neither pre-computes similarity measures nor maintain an index of seen comparisons. Unsurprisingly, the total retrieval time increases linearly as the number of queries increases. Figure 5 shows how the average time of retrieval for each query reduces over time providing additional insight into the advantage of caching over non-caching. The average time per query remains consistently high for Non-Cached. In contrast, the average time per query significantly reduces over time for Cached. By the 100th query, Cache was already more than 2 times better than Non-Cached.

Similarity results were obtained for ontology feature-based similarity. In general, Pre-loaded option is useful if most of the ontology nodes are being used in the casebase. Cached is an efficient approach if queries often repeat and for large

scale ontologies where only a few nodes are used. Non-Cached can be useful for smaller rapidly changing ontologies.

6 Conclusion

In this paper, we discussed how architectural challenges influenced decisions on how to include semantic similarity measures for case retrieval in a microservices architecture. We showed how the similarity measures can be implemented on the CLOOD framework and presented an experimental evaluation to demonstrate the impact of different options for using ontology-based semantic similarity measures.

Future work will provide support for additional methods of computing the ontology-based similarity measures including approaches for embedding knowledge graphs. As a supplement to the open-sourced and extensible CLOOD framework, the team is currently building the first CBR-as-a-Service platform. CLOOD API will be a software-as-a-service offering where organisations, developers and researchers can harness the power CBR and CLOOD within few minutes. The key reason for this direction is due to the time consumed for setting up CBR frameworks for each project/product. This can be solved by using a CBR-as-a-Service platform where the entire CBR system is itself a single microservice. Infrastructure, security, maintenance and bug-fixes can be centrally handled reducing any overhead for organisations to adapt and integrate CBR into their applications. This will be a solution for small-medium scale businesses with limited expertise on CBR and cloud to setup the systems. With an interactive and simple dashboard the entire CBR cycle can be ready and exposed as API endpoints in very few steps. You can submit your interest in this platform at `https://cloodcbr.com`.

References

1. Amailef, K., Lu, J.: Ontology-supported case-based reasoning approach for intelligent m-government emergency response services. Decision Support Systems **55**(1), 79–97 (2013)
2. Amin, K., Lancaster, G., Kapetanakis, S., Althoff, K.D., Dengel, A., Petridis, M.: Advanced similarity measures using word embeddings and siamese networks in cbr. In: Proceedings of SAI Intelligent Systems Conference. pp. 449–462. Springer (2019)
3. Cer, D., Yang, Y., Kong, S.y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al.: Universal sentence encoder. arXiv preprint arXiv:1803.11175 (2018)
4. Díaz-Agudo, B., González-Calero, P.A., Recio-García, J.A., Sánchez-Ruiz-Granados, A.A.: Building cbr systems with jcolibri. Science of Computer Programming **69**(1-3), 68–75 (2007)
5. Ghofrani, J., Lübke, D.: Challenges of microservices architecture: A survey on the state of the practice. ZEUS **2018**, 1–8 (2018)

6. González-Calero, P.A., Díaz-Agudo, B., Gómez-Albarrán, M., et al.: Applying dls for retrieval in case-based reasoning. In: In Procs. of the 1999 Description Logics Workshop (Dl'99). Linkopings universitet. Citeseer (1999)
7. Hliaoutakis, A., Varelas, G., Voutsakis, E., Petrakis, E.G., Milios, E.: Information retrieval by semantic similarity. International journal on semantic Web and information systems (IJSWIS) **2**(3), 55–73 (2006)
8. Khattak, F.K., Jeblee, S., Pou-Prom, C., Abdalla, M., Meaney, C., Rudzicz, F.: A survey of word embeddings for clinical text. Journal of Biomedical Informatics **100**, 100057 (2019)
9. Lastra-Díaz, J.J., Goikoetxea, J., Taieb, M.A.H., García-Serrano, A., Aouicha, M.B., Agirre, E.: A reproducible survey on word embeddings and ontology-based methods for word similarity: linear combinations outperform the state of the art. Engineering Applications of Artificial Intelligence **85**, 645–665 (2019)
10. Lin, D., et al.: An information-theoretic definition of similarity. In: ICML. vol. 98, pp. 296–304 (1998)
11. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems **26** (2013)
12. Montero-Jiménez, J.J., Vingerhoeds, R., Grabot, B.: Enhancing predictive maintenance architecture process by using ontology-enabled case-based reasoning. In: 2021 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–8. IEEE (2021)
13. Nkisi-Orji, I., Wiratunga, N., Palihawadana, C., Recio-García, J.A., Corsar, D.: Clood cbr: Towards microservices oriented case-based reasoning. In: International Conference on Case-Based Reasoning. pp. 129–143. Springer (2020)
14. Qin, Y., Lu, W., Qi, Q., Liu, X., Huang, M., Scott, P.J., Jiang, X.: Towards an ontology-supported case-based reasoning approach for computer-aided tolerance specification. Knowledge-Based Systems **141**, 129–147 (2018)
15. Rada, R., Mili, H., Bicknell, E., Blettner, M.: Development and application of a metric on semantic nets. IEEE transactions on systems, man, and cybernetics **19**(1), 17–30 (1989)
16. Recio-Garía, J.A., Díaz-Agudo, B.: Ontology based cbr with jcolibri. In: International Conference on Innovative Techniques and Applications of Artificial Intelligence. pp. 149–162. Springer (2006)
17. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. arXiv preprint cmp-lg/9511007 (1995)
18. Sánchez, D., Batet, M., Isern, D., Valls, A.: Ontology-based semantic similarity: A new feature-based approach. Expert systems with applications **39**(9), 7718–7728 (2012)
19. Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N.J., Popa, R.A., Gonzalez, J.E., Stoica, I., Patterson, D.A.: What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM **64**(5), 76–84 (2021)
20. Taibi, D., El Ioini, N., Pahl, C., Niederkofler, J.R.S.: Patterns for serverless functions (function-as-a-service): A multivocal literature review (2020)
21. Wu, Z., Palmer, M.: Verb semantics and lexical selection. arXiv preprint cmp-lg/9406033 (1994)
22. Xie, R., Tang, Q., Qiao, S., Zhu, H., Yu, F.R., Huang, T.: When serverless computing meets edge computing: architecture, challenges, and open issues. IEEE Wireless Communications **28**(5), 126–133 (2021)