IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 28, 2021, accepted April 27, 2021, date of publication April 30, 2021, date of current version May 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3076833

Building Real-Time Ontology Based on Adaptive Filter for Multi-Domain Knowledge Organization

JIANHUI ZHOU, XIAOXIA SONG, YONG LI, YUN GAO, AND XULONG ZHANG
School of Computer and Network Engineering, Shanxi Datong University, Datong 037009, China

Corresponding author: Xiaoxia Song (sxxly2002@163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1701703, in part by the Natural Science Foundation of Shanxi Province under Grant 201901D111311, and in part by the Science and Technology Program of Datong City under Grant 2019165.

ABSTRACT Multi-domain knowledge organization is an effective way of correlating cross-domain knowledge or intercommunicating between cross-domain knowledge systems. As a knowledge organization model, ontology is widely used in information and management systems. To organize multi-domain knowledge, ontologies in different domains correlate to each other directly or indirectly. Generally, matching and integrating ontologies of different domain into a large scale ontology is the common way of directly correlating, while building a top level ontology is the main method for indirectly correlating. As the scale of domain ontologies get larger and larger, both direct and indirect methods become more difficult and time-consuming. In order to improve the organization of multi-domain knowledge, this paper presents a novel ontology organization method to build real-time ontology by adaptive filter while user presenting requirements. Only the entities related to user requirements are integrated, while building a real-time ontology. Firstly, the method searches domain ontologies that are related to user requirements. Then sub-ontologies are extracted from search results by filter, and they are integrated into a new ontology under direction of filter, i.e. real-time ontology. Finally, four criteria are introduced to evaluate real-time ontology. The experiment results illuminate that real-time ontology perform excellently in accuracy, recall, correctness and especially time-consuming.

INDEX TERMS Real-time ontology, multi-domain knowledge organization, ontology matching, ontology integration, knowledge engineering.

I. INTRODUCTION

Knowledge organization is the important research topic in knowledge engineering. As a knowledge organization model, ontology defines domain concepts and their relationships, which can effectively describe semantic information among concepts [1]. With the rapidly increasing of knowledge and the development of systems engineering, scale of ontology is becoming larger and larger, and the requirement of organizing multi-domain knowledge is increasing. Especially in system of developing complex product (such as aircraft, vehicle, watercraft, robot, etc.) [2], multi-domain knowledge organization is a fundamental tool, which can improve the effectiveness and efficiency of knowledge management reasonably, (such as knowledge retrieval, knowledge application, knowledge sharing etc.).

The associate editor coordinating the review of this manuscript and approving it for publication was Feng Xia.

There are two typical methods of organizing multi-domain knowledge by ontology. One is to integrate each domain ontology into an ontology who has lots and lots of concepts and relationships [3]. This integrated ontology was named as super-large ontology (SLO) in this paper. The other is to build an ontology who has higher level concepts of each domain, named as top-level ontology (TLO), by which, domain ontologies can be correlated to each other [4].

Integrating each domain ontology into SLO is a very time-consuming process, because the scale of SLO is too large. Furthermore, the larger the scale of SLO is, the lower accuracy and recall rate of matching ontologies are, and ontology matching is the key method of generating SLO.

Building TLO is a manual process normally. This process requires the participation of multiple domain experts. Maintaining and updating TLO is difficult and high-cost particularly. To organize multi-domain knowledge, TLO and the domain ontologies are applied together. Instantaneity of

application is low, because TLO and each domain ontology need to be matched at the time of application.

To further improve multi-domain knowledge organization and avoid the disadvantages of SLO and TLO, this paper presents a novel ontology model and a method of building it, whose name is real-time ontology (RTO). At the time of application, RTO is generated automatically according to the task. In this process, the entities related to the task are selected, and then they are integrated into RTO. RTO has higher instantaneity, higher correctness, acceptable completeness, and it is low-cost particularly.

The main contributions made in this paper are as follows:

- The novel ontology model, i.e. RTO is presented for multi-domain knowledge organization.
- The method of building RTO based on adaptive filter is presented and verified.
- The relevancy decay function and adaptive threshold are presented to improve the method.

The rest of the paper is organized as follows: Section II describes the related works; Section III clarifies the knowledge organization models including SLO and TLO, and presents RTO; Section IV elaborates the process of building RTO based on adaptive filter; Section V shows the experimental results, and evaluates the RTO; finally, Section VI draws the conclusions.

II. RELATED WORKS

In the process of building RTO, technologies such as ontology search, sub-ontology extraction, ontology matching and integration were implemented successively. In this section, the research review of each technology is elaborated as follows.

A. ONTOLOGY SEARCH

As a semantic ontology search system, Swoolge contained variety patterns of search configuration [5]. In general pattern, feature vector was used to formalize ontologies. Swoolge adopted the traditional inverted index model to index the entities in ontology. Scores of each ontology in query result set were determined by calculating TF/IDF of feature vector in inverted index model [6]. OntoKhoj was the API whose functions included building ontology, searching ontology, visualizing ontology, and matching ontology [7]. In the function of searching ontology, OntoKhoj took Rainbow Tool as classifier, by which the descriptive texts of ontology were classified to several categories [8]. OntoKhoj calculated the similarities between search keywords and categories to determine the scores of each ontology. OntoSearch was an ontology search system mainly for the ontology resource in web [9]. It deployed the core components of Google search engine. OntoSearch described ontology as a set of 3-tuples, and saved all terms of 3-tuples. With matching the terms and search keywords, target 3-tuples were return to user. In OntoSearch, the search results, i.e. target 3-tuples were merged by logical rules and visualized as a graphic comprehensibly [10]–[12]. On this basis, researchers developed OntoSearch2 which improved some functions of OntoSearch [13]. Distributed memory was utilized in OntoSearch2 to enhance the efficiency of saving 3-tuples [14]. Class and Individual in ontology were saved in different databases. OntoSearch2 introduced fuzzy DL-Lite to reason 3-tuples of ontology, and interacted with external semantic resource [15]–[18]. SQORE was another ontology search engine based on 3-tuple. In SQORE [19], external semantic resource was introduced as references. Using the knowledge in references, the set of search keywords were extended to a new superset who had more search keywords. SQORE conversed search keywords and ontology to a set of 3-tuples, and SQORE proposed a method of calculating the similarity between 3-tuples. Scores of searching ontology were determined by calculating the similarity between the 3-tuples of keywords and the 3-tuples of ontologies [20]–[22].

In general, indexing ontology was efficient to search ontology. But in existed methods or systems, only URI, labels, described text were indexing, which resulted in that the accuracy and recall rate of methods were low. In this paper, to improve the performance of searching ontology, more information of entities in ontology were indexed for searching ontology, such as its type, depth, number of relationships, and frequency of labels.

B. SUB-ONTOLOGY EXTRACTION

Extracting sub-ontology is the approach to deal with large-scale ontology and acquire key information in ontology [23]–[25]. It eliminates lots of entities in ontology that are irrelevant to the user requirements. Bhatt proposed a distributed method of extracting sub-ontology [26]. According to user requirements, domain experts collaboratively annotated entities in ontology to required entities or not required entities. The required entities were regarded as vertexes. Edges between vertexes were generated by the algorithm proposed by Bhatt. The result of integrating vertexes and edges was the extracted sub-ontology. In [27], a grid application service framework for extracting sub-ontology was proposed, which consisted of the client, server, and grid layers. The process of extracting sub-ontology included two phases. In separation phase, user annotated required node, not required node, and redundant node on client layer. In optimization phase, sub-ontology extraction mechanism removed, inserted, and moved entities in turn on server layer. The results and the based ontology were stored on grid layer. Flahive proposed Ontology as a Service (OaaS) [28], in which ontology was reused and extracted as a service. This work identified four sub-ontology operations whose names were extend, add, merge, and replace [29]–[32]. In each operation, sub-ontology extraction was necessary. The algorithm of sub-ontology extraction was presented in [33], it had two method: maximum extraction method and minimum extraction method. The maximum extraction method aimed to extract as many entities from the source ontology as possible in the initial stage of removing requirements to revisit the

source ontology. The minimum extraction method aimed to extract as few entities as possible in the initial stage but may require further services from the cloud. In these two methods, entities in the source ontology were labeled to selected, deselected, and undecided firstly, and then entities were selected or deleted by the consistency rule they made.

As mentioned above, the existed methods of extracting sub-ontology generally needed domain experts or users to annotate the entities initially. But the process of building RTO is accomplished automatically. Refer to our previous work [34], and an adaptive filter is used in this paper to eliminate the entities that irrelevant to user requirements. Based on adaptive filter, building RTO automatically can be realized.

C. ONTOLOGY MATCHING AND INTEGRATION

The essence of ontology matching and integration is to generate alignment between ontologies [35]. According to alignment, ontologies can be integrated into a new ontology. Agreement Maker Light (AML) was an automated ontology matching system that was developed with both extensibility and efficiency in mind [36], [37]. AML focused on the lexical similarities between entities, and concerns with efficiency. It was suitable for dealing with large-scale ontology matching problem. ALIN was an ontology matching system, which generate alignment interactively [38]. ALIN utilized WordNet as the external resource, and took linguistic matching technology to calculate similarity between entities. Initial correspondences were generated automatically, and then interactions were made with the experts to modify correspondences. As an ontology matching system, LogMap [39], [40] is efficient and scalable, and it utilized several kinds of elements for its scalability, which include lexical indexation, logic-based module extraction, propositional Horn reasoning, axiom tracking, local repair, and semantic indexation. In matching process, sophisticated reasoning and repair techniques were deployed to minimize the number of logical inconsistencies. POMap consisted of semantic matcher, syntactic matcher, and structural matcher [41]. In the initial phase, ontologies were indexed and preprocessed to input POMap, and then POMap implemented semantic matcher and syntactic matcher sequentially to find correspondences between entities. On the basis of these correspondences, structural matcher was implemented to extend them. Lily utilized several ontology matching techniques to facilitate alignment [42]. It deployed specific algorithms to match ontology, which included Generic Ontology Matching, Large-scale Ontology Matching, and Instance Ontology Matching. Alignments were improved and verified by ontology matching debugging and tuning in Lily. As the main technique in Lily, Semantic sub-graph was developed to capture the real meanings of ontology entities, and reduce the negative effects of the matching uncertainty.

In process of building RTO, sub-ontologies were matched and integrated into a new ontology, i.e. RTO. The scale of these sub-ontologies was small. To satisfy the features of RTO, appropriate method of ontology matching and integration should be automatic, efficient, and precise. According to results of OAEI 2020 [43], AML was adopted in this paper, who was an automatic method and low consumption, and whose accuracy and recall were high enough.

III. PRELIMINARIES

A. DOMAIN ONTOLOGY

An ontology is represented as a set of entities who contains concepts, relationships, and individuals. It can be denoted as (C, R, I), where C, R, and I are the sets of concepts, relationships, and individuals, respectively. If entities in ontology have common discourse domain, the ontology is named as domain ontology. It can be denoted as follows:

$$\mathrm{DO} = \{C, R, I | C, R, I \in \mathrm{D}\} \tag{1}$$

where DO is a domain ontology, D is the discourse domain.

B. SUPER-LARGE ONTOLOGY (SLO)

SLO is an ontology that integrates multiple domain ontologies. The mainly technologies of generating SLO are ontology matching and ontology integration. It is generally an automatic or semi-automatic process. SLO application in multi-domain knowledge systems is shown in Fig. 1. SLO can be denoted as follows:

$$\mathrm{SLO} = \mathrm{DO}_1 \otimes \mathrm{DO}_2 \otimes \cdots \otimes \mathrm{DO}_n \tag{2}$$

where DO_i are domain ontologies who belong different discourse domains, and $\otimes$ means the integrated operation.

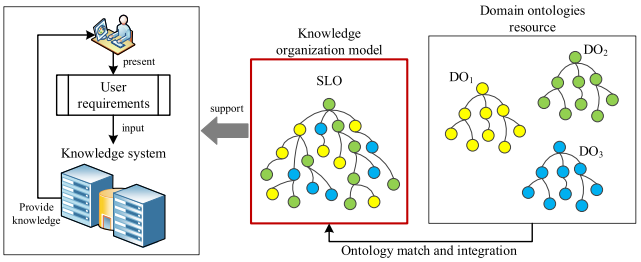


FIGURE 1. Organizing knowledge by SLO and its application.

C. TOP-LEVEL ONTOLOGY (TLO)

TLO is an ontology whose entities contain the interdisciplinary core concepts. Building TLO is generally a manual process, which is accomplished by domain experts. TLO can be regarded as the bridge that connects each domain ontologies. As shown in Fig. 2, TLO and domain ontologies are applied together in multi-domain knowledge systems. For simplicity, in this paper, TLO and domain ontologies are considered as a whole object, who is denoted as follows:

$$\mathrm{TLO} = \{\mathrm{TLO}, \mathrm{DO}_1, \mathrm{DO}_2, \cdots, \mathrm{DO}_n\} \tag{3}$$

where DO_i are domain ontologies who belong different discourse domains.

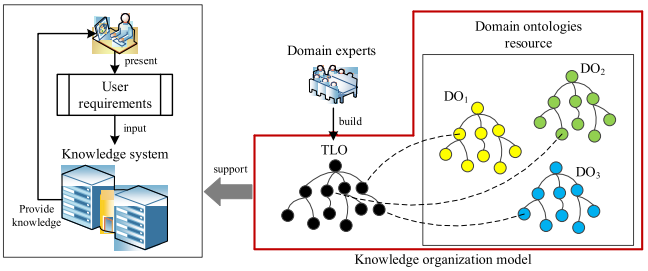


FIGURE 2. Organizing knowledge by TLO and its application.

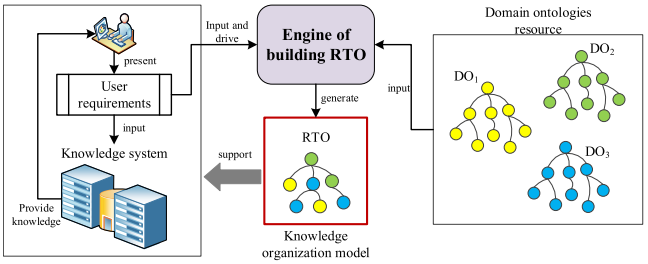


FIGURE 3. Organizing knowledge by RTO and its application.

D. REAL-TIME ONTOLOGY (RTO)

RTO is the ontology model proposed in this paper. As shown in Fig. 3, unlike SLO and TLO, user requirements are the input of building RTO, and RTO is an ontology who satisfies user requirements and is lightweight. It only retains the entities in domain ontologies who are related to user requirements. In each domain ontology, the entities satisfying user requirements are the sub-set of it. This sub-set of domain ontology are named as sub-ontology in this paper. RTO is the result of integrating sub-ontologies extracted from multiple domain ontologies. Compare with SLO and TLO, the advantages of RTO are summarized in Tab. 1, and RTO is denoted as follows:

$$\begin{cases} \text{RTO} = sub(\text{DO}_1) \otimes sub(\text{DO}_2) \otimes \cdots \otimes sub(\text{DO}_n) \\ sub(\text{DO}) = \{C_S, R_S, I_S | C_S \subseteq C, R_S \subseteq R, I_S \subseteq I\} \end{cases} \tag{4}$$

where C_S, R_S, and I_S are sub-set of C, R, and I in DO, respectively.

IV. METHOD OF BUILDING RTO BASED ON ADAPTIVE FILTER

RTO is built at the time of user presenting requirements. User can present requirements in several ways, such as inputting text, selecting work nodes in workflow, designing product online, analyzing data in workspace, etc. These presented requirements are transformed to a set of domain terms that is input of building RTO, but the process of which is not focused on in this paper. So, for simplicity, user requirements in this paper are regarded as a set of domain terms, which is denoted as follows:

$$\text{T} = \{t_1, t_2, \cdots, t_n\} \tag{5}$$

where t_i are domain terms, T is user requirements, i.e. inputs of building RTO.

TABLE 1. Building and application feature of SLO, TLO, and RTO.

knowledge organization model	SLO	TLO	RTO
build time	before application	before application	at the time of application
build manner	automatic	manual	automatic
consumption	high	-	low
difficulty of update	great	great	no need update
efficiency of application	low	low	high

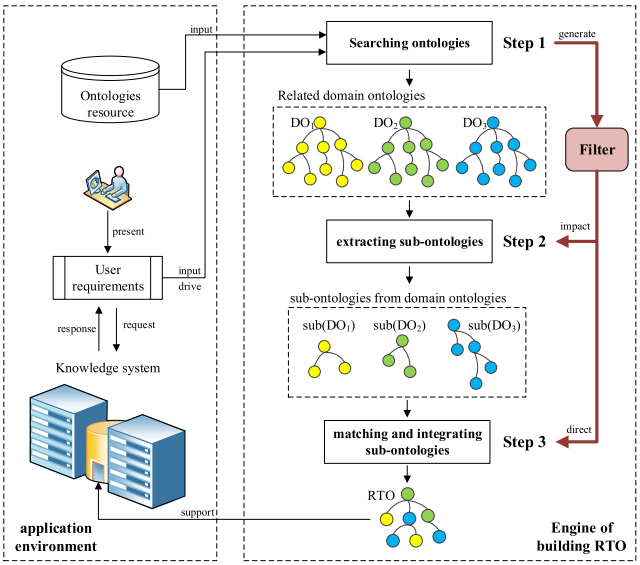


FIGURE 4. Framework of building RTO based on adaptive filter.

The framework of building RTO is illustrated in Fig. 4. As shown in it, the process includes three key steps:

- *Step 1:* Searching related domain ontologies, and generating filter to impact or direct the subsequent steps.
- *Step 2:* Extracting sub-ontologies from the related domain ontologies by filter.
- *Step 3:* Matching the sub-ontologies, and merging them into RTO under the direction of filter.

The filter mentioned above is a set of 4-tuples. A 4-tuple contains the URI of related domain ontology, degree of importance of related domain ontology (DoI), relevancy decay function (f-decay), and the threshold of relevancy (δ). The filter is denoted as follows:

$$\begin{cases} \text{Filter} = \{\text{F}(\text{DO}_1), \text{F}(\text{DO}_2), \cdots, \text{F}(\text{DO}_n)\} \\ \text{F}(\text{DO}_i) = (\text{URI}_i, \text{DoI}_i, f\text{-decay}_i, \delta_i) \end{cases} \tag{6}$$

DoI equals to the score of domain ontology which is calculated in Step 1, and f-decay is determined by DoI.

The pseudo-code of the framework is introduced in Algorithm 1. Each step will be elaborated in the subsequent sections.

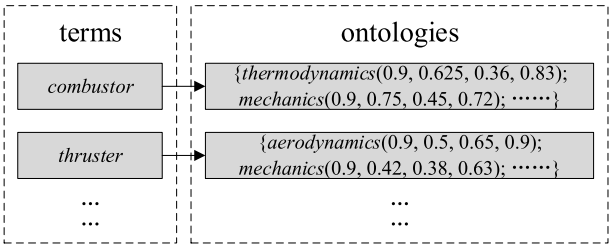


FIGURE 5. Example of domain ontology index.

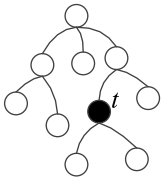


FIGURE 6. Example of entity depth in ontology.

A. STEP 1: SEARCHING RELATED DOMAIN ONTOLOGIES

The target of this step is to search the domain ontologies which are related to the set of domain terms and generate filter. It is the key element that calculating the scores of related domain ontologies. Domain ontologies were indexed to improve the efficiency of this step, in which the weights of entities in domain ontology were calculated.

To calculate the weights of entities in ontology reasonably, four kinds of information about entities were utilized, which are type, depth, number of relationships, and frequency of labels. For simplicity, in this paper, the weights of them are denoted as WoT, WoD, WoR, and WoF, respectively. As shown in Fig. 5, e.g., *combustor* was an entity in domain ontology namely *thermodynamics*, and 0.9, 0.625, 0.36, 0.83 were the WoT, WoD, WoR, and WoF values of *combustor* in *thermodynamics*, respectively.

In domain ontology, there are three types of entities, namely concept, relationship, and individual. Generally, the higher the level of abstraction is, the better the generalization ability is, and the more important the entities are. So, the importance of concept > relationship > individual, and WoT is valued as formula (7). 0.5, 0.6, 0.9 are determined by experience.

$$\mathrm{WoT}(t) = \begin{cases} 0.5 & \text{if type of } t \text{ is } I \\ 0.6 & \text{if type of } t \text{ is } P \\ 0.9 & \text{if type of } t \text{ is } C \end{cases} \tag{7}$$

The depth of entities reflects their scope of description in domain ontology. The lower the depth is, the more representative the entities are, and the more importance they are. WoD is calculated by formula (8).

$$\mathrm{WoD}(t) = \frac{\mathrm{depth}_{\max} - \mathrm{depth}(t)}{\mathrm{depth}_{\max}} \tag{8}$$

where t is label of an entity; $\mathrm{depth}(t)$ is the depth of t in ontology; and $\mathrm{depth}_{\max}$ is the maximum depth in ontology, e.g., as shown in Fig. 6, $\mathrm{depth}(t)$ is 3, and $\mathrm{depth}_{\max}$ is 4.

In domain ontology, if an entity had a good deal of relationships to other entities, it is very important for the ontology. According to this idea, WoR is calculated by formula (9).

$$\mathrm{WoR}(t) = \mathrm{r}(t) \times \sqrt{\left(\Sigma_{i=0}^{n} \mathrm{r}(t_i)^2\right) \Big/ n} \tag{9}$$

where t is label of an entity; $\mathrm{r}(t)$ is the number of relationships t has; t_is are all entities in ontology; and n is the number of all entities in ontology.

In domain ontology, usually, the entities have some descriptive text to specify itself, i.e. label of one entity may be presented in descriptive texts of other entities. The frequency of label of entity in descriptive texts can directly reflects its importance to domain ontology. WoF was calculated by formula (10), who referred to TF/IDF of words.

$$\mathrm{WoF}(t) = \frac{\mathrm{N}(t)}{\mathrm{N}} \times \log \frac{\mathrm{N}(C)}{\mathrm{N}(C_t)} \tag{10}$$

where t is label of an entity; $\mathrm{N}(t)$ is the number of t in all descriptive texts; N is the number of all terms in all descriptive texts; $\mathrm{N}(C)$ is the number of all entities in ontology; and $\mathrm{N}(C_t)$ is the number of entities whose descriptive texts contain t in ontology.

Notice that, WoR and WoF may bigger than 1. They are normalized to 0-1. But this operation is ignored in formulas and algorithms.

The comprehensive weight of t is finally determined by formula (11).

$$\mathrm{W}(t) = w_1 \mathrm{WoT}(t) + w_2 \mathrm{WoD}(t) + w_3 \mathrm{WoR}(t) + w_4 \mathrm{WoF}(t) \tag{11}$$

where w_1, w_2, w_3, w_4 are weights of each items. They are determined by experience generally.

In search phase, the score of related domain ontology is calculated by formula (12).

$$\mathrm{score}(\mathrm{DO}) = \frac{1}{n} \sum_{i=1}^{n} W(t_i) \tag{12}$$

where t_is are the terms that domain ontology corresponds in index.

The pseudo-codes of indexing and searching domain ontologies are introduced in Algorithm 1 and Algorithm 2, respectively.

B. GENERATING FILTER ADAPTIVELY

If the score of domain ontology was higher, the degree of its relevancy to user requirements was higher, and it had more entities related to user requirements. That is, the scale of sub-ontology extracted from it should be little larger. Therefore, the core idea of adaptive filter is that the higher the score of domain ontology is, the slower the decay speed of relevancy is, and the lower the threshold of relevancy is.

Accordingly, f-decay and δ in filter are calculated by formula (13) and (14), respectively. Variable k means the k^{th}

Algorithm 1 Indexing Domain Ontologies

Input: domain ontologies (List < Onto > *DO*)
Output: index of domain ontologies (HashMap < String, List > *index*)
/*This algorithm is executed before searching domain ontologies*/
HashMap < String, List > *index* //saving the index results
for $i = 0$ to length(*DO*):
for each entity e in $DO[i]$:
$term = e$.label //label is regarded as the name of entity
$onto = DO[i]$.URI //URI is the unique ID of ontology
calculate *WoT*, *WoD*, *WoR*, *WoF* of e by (7), (8), (9), (10)
index.key = *term*
index.value.add(*onto*)
index.value.add(*WoT*)
index.value.add(*WoD*)
index.value.add(*WoR*)
index.value.add(*WoF*)
end for
end for
return *index*

iteration in the process of extracting sub-ontology, and it will be elaborated in next sections.

$$f - decay_i(k) = \frac{1 + \mathrm{DoI}_i^{-k}}{1 + \mathrm{DoI}_i^{k}} \tag{13}$$

$$\delta_i = \exp(-\frac{\mathrm{DoI}_i}{\sum_{i=1}^{n} \mathrm{DoI}_i}) \tag{14}$$

After the filter was generated, f-decay and δ will be utilized to impact the process of extracting sub-ontologies in Step 2. DoI will be utilized to direct the process of merging sub-ontologies in Step 3.

C. STEP 2: EXTRACTING SUB-ONTOLOGIES BY FILTER

In the previous section, related domain ontologies user searched. From which, sub-ontologies are extracted in this step.

The target of this step is to find the entities related user requirements in domain ontologies. In one domain ontology, the entities whose label existed in user requirements are initialized to start entities, which are added into related set, and the other entities that have relationships to start entities are added into candidate set. Entities in candidate set are moved into related set if the relevancy of between them and entities in related set is greater than threshold, and they are taken as new start entities. Related set is updated by performing the above operations iteratively until it does not change anymore, and final related set is the extracted sub-ontology. The pseudo-code of this step is introduced in Algorithm 3. Generally, in the k^{th} iteration, the relevancy between entities is calculated by (15).

$$\mathrm{R}(e_{\mathrm{R}}, e_{\mathrm{C}}; k, \mathrm{DO}_i) = sim(e_1, e_2) \times f - decay_i(k) \tag{15}$$

Algorithm 2 Searching Related Domain Ontologies

Input: user requirements (List < String > *T*);
index of domain ontologies (HashMap < String, List > *index*)
Output: a set of related domain ontologies (List < Onto > *relatedDO*)
List < Onto > *relatedDO* //saving the search results
HashMap < String, Double > *ontoScore*
//saving the scores of related domain ontologies
Double *w*1, *w*2, *w*3, *w*4
for $i = 0$ to length(*T*):
for each item x in *index*:
if $T[i] == x$.key:
String $onto = x$.key
Double $score = w1^{*}x$.value.get(*WoT*) +
$w2^{*}x$.value.get(*WoD*) +
$w3^{*}x$.value.get(*WoR*) +
$w4^{*}x$.value.get(*WoF*)
ontoScore.key = *onto*
ontoScore.value = *score* + *ontoScore*.value
end if
end for
end for
for each item o in *ontoScore*:
Onto O = getOnto(o.key) //Ontology can be obtained by its URI
relatedDO.add(O)
end for
return *relatedDO*

where e_{R} and e_{C} are entity in related set and candidate set, respectively; $\mathrm{R}(e_{\mathrm{R}}, e_{\mathrm{C}}; k, \mathrm{DO}_i)$ is the relevancy between e_{R} and e_{C} in DO_i at the k^{th} iteration; $sim(e_{\mathrm{R}}, e_{\mathrm{C}})$ is the similarity between e_{R} and e_{C}, and $sim(e_{\mathrm{R}}, e_{\mathrm{C}})$ was introduced in our previous work. If $\mathrm{R}(e_{\mathrm{R}}, e_{\mathrm{C}}; k, \mathrm{DO}_i)$ is greater than δ_i in the filter, e_{C} is added into related set.

D. STEP 3: INTEGRATING SUB-ONTOLOGIES UNDER DIRECTION OF FILTER

In the previous section, a set of sub-ontologies was extracted from the domain ontologies that were related to user requirements. In this step, the sub-ontologies are matched firstly to generate alignments between each pair of them, and then according to these alignments, the sub-ontologies are integrated into a new ontology, i.e. RTO.

In the first phase, an automated ontology matching method namely Agreement Maker Light (AML) [36] is deployed to match sub-ontologies. The time-consumption of AML is low, and the accuracy and recall rate of AML are great enough. It is suitable for building RTO.

In the second phase, as shown in Fig. 7, alignments are merged into a global alignment, and the entities that exist in the global alignment are merged into a new ontology, called the skeleton of RTO; and then, the other entities are added

Algorithm 3 Extracting Sub-Ontologies by Filter

Input: a set of related domain ontologies (List < Onto > *relatedDO*);
user requirements (List < String > T);
Filter (List < 4-tuple > *filter*)
Output: a set of sub-ontologies (List < Onto > *sub_DO*)

```
List < entity > relatedSet
List < entity > candidateSet
List < Onto > sub_DO
for i = 0 to length(relatedDO):
  for each entity e in relatedDO[i]:
    if e.label exists in T:
      relatedSet.add(e) //initialize relatedSet
      if e has relationship to entity ec in relatedDO[i]:
        candidateSet.add(ec) //initialize candidateSet
      end if
    end if
  end for
  while relatedSet != NULL:
   k = 1
   for each entity e in relatedSet:
    for each entity ec in candidateSet
      if R(e, ec, k, relatedDO[i]) > filter[i].threshold:
        //R is calculated by (15)
        relatedSet.add(ec)
        candidateSet.remove(ec)
      end if
    end for
    sub_DO[i].addEntity(e)
    relatedSet.remove(e)
   end for
   k = k+ 1
  end while
 end for
return sub_DO
```

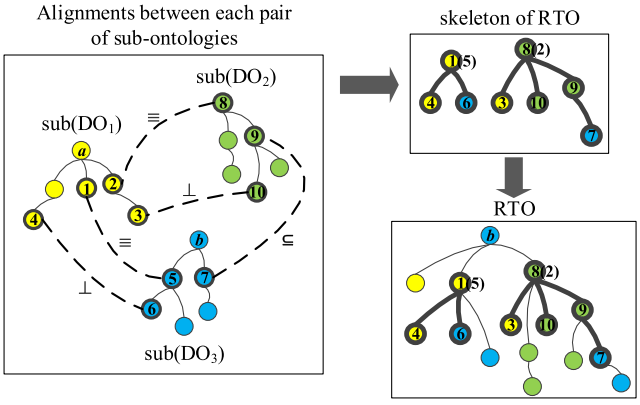


FIGURE 7. The process of merging sub-ontologies.

into the skeleton of RTO in proper order, the result of which is RTO.

There are three kinds of correspondence in alignment, that are equivalence (≡), subsumption (⊆), and disjointness (⊥).

To merge the entities into the skeleton of RTO, three rules are regulated as follows:

- IF A ≡ B, THEN merge A and B.
- IF A ⊆ B, THEN A is the sub-entity of B.
- IF A ⊥ B, THEN A and B have the common super-entity.

Obviously, confliction exists in the process of merging entities ineluctably. As shown in Fig. 6, e.g., entity 3 is the sub-class of entity 2, entity 10 is the sub-sub-class of entity 8, but, entity 2 equals to entity 8, entity 3 is disjoint to entity 8. In the filter, if DoI_1 is greater than DoI_2, the relationship in sub(DO_1) is maintained preferentially. So, entity 10 is modified to the sub-class of entity 8 in the skeleton of RTO.

After the skeleton of RTO is generated, the other entities in sub-ontologies are added into the skeleton by recovering their relationships. Similarly, confliction exists in process all the same. It is dealt with according to the filter too. As shown in Fig. 6, e.g., entity *a* is the super-class of entity 1, and entity *b* is the super-class of entity 5, but, entity 1 equals to entity 5. If DoI_3 is greater than DoI_1, entity *b* is selected preferentially in the RTO.

The pseudo-code of integrating sub-ontologies into RTO is introduced in Algorithm 4.

V. EVALUATION

A. EXPERIMENTAL PREPARATION

To evaluate the proposed method of building RTO, we built 10 domain ontologies about robot under the guidance of WordNet2.0, that is a lexical database for English. The domains of ontologies are different with each other. The entities in 10 domain ontologies were collected from WordNet2.0, and the details of domain ontologies is shown in Tab. 2.

TABLE 2. Instances of domain ontologies about robot.

DO	Domain	Number of concepts	Number of relationships
DO_1	Mechanics	1027	3859
DO_2	Structure	952	4142
DO_3	Electronics	493	1825
DO_4	Cybernetics	917	3581
DO_5	Dynamics	826	3479
DO_6	Bionics	1840	5571
DO_7	Telecommunications	884	2950
DO_8	Computer Science	1129	3510
DO_9	Materials Science	1359	3847
DO_{10}	Artificial Intelligence	586	1865

As the inputs of experiment, we designed 9 sets of terms. The sizes of each set were 2-10, and the terms in one set came from different domains. For each set of terms, we collected the related words from WordNet2.0 as the references

TABLE 3. Lists of domain ontologies for each set of terms.

Inputs	Domains ontologies that contained terms of T_i
T_1	DO_3, DO_4
T_2	DO_1, DO_2, DO_5
T_3	DO_1, DO_5, DO_6, DO_8
T_4	DO_3, DO_6, DO_7, DO_9, DO_{10}
T_5	DO_1, DO_3, DO_4, DO_5, DO_6, DO_9
T_6	DO_2, DO_4, DO_5, DO_6, DO_7, DO_8, DO_{10}
T_7	DO_1, DO_3, DO_4, DO_5, DO_7, DO_8, DO_9, DO_{10}
T_8	DO_1, DO_2, DO_3, DO_4, DO_6, DO_7, DO_8, DO_9, DO_{10}
T_9	DO_1, DO_2, DO_3, DO_4, DO_5, DO_6, DO_7, DO_8, DO_9, DO_{10}

of experiment. The domain ontologies that contained terms in each set are listed in Tab. 3.

B. EVALUATION CRITERIA

In this paper, we introduce the evaluation criteria of the method of building RTO. They are *accuracy*, *recall*, *correctness*, and *run-time*.

Accuracy and *recall* are utilized to evaluate the effectiveness of the method of building RTO. They are calculated by formula (16), and (17), respectively.

$$accuracy = \frac{\mathrm{N}(C_{\mathrm{RTO}} \cap C_{\mathrm{ref}})}{\mathrm{N}(C_{\mathrm{RTO}})} \tag{16}$$

$$recall = \frac{\mathrm{N}(C_{\mathrm{RTO}} \cap C_{\mathrm{ref}})}{\mathrm{N}(C_{\mathrm{ref}})} \tag{17}$$

where $\mathrm{N}(C_{\mathrm{RTO}} \cap C_{\mathrm{ref}})$ is the number of the concepts that both RTO and reference have; $\mathrm{N}(C_{\mathrm{RTO}})$ is the number of all concepts in RTO; and $\mathrm{N}(C_{\mathrm{ref}})$ is the number of all concepts in reference.

Correctness is utilized to evaluate the quality of the RTO. It is calculated by formula (18).

$$correctness = \frac{\mathrm{N}(R_{\mathrm{correct}})}{\mathrm{N}(R_{\mathrm{RTO}})} \tag{18}$$

where $\mathrm{N}(R_{\mathrm{correct}})$ is the number of the relationships in RTO that were consistent to the relationships in reference; $\mathrm{N}(R_{\mathrm{RTO}})$ is the number of all relationships in RTO.

Run-time is no need to be calculated, it is tested by system directly. *Run-time* is utilized to evaluate the efficiency of the method of building RTO.

C. EVALUATION RESULTS

We tested the method of building RTO by each input. As shown in Tab. 4, RTO[T_i] means the RTO with the T_i as its input. We visualized partial data in Tab. 4 as Fig. 8. It is indicated that *accuracy* and *recall* decreased gradually as the value of *i* increase. Nevertheless, they did not fluctuate much. *Correctness* almost stayed constant. So, in aspect of the

Algorithm 4 Integrating Sub-Ontologies into RTO

Input: a set of sub-ontologies (List < Onto > *sub_DO*);
Filter (List < 4-tuple > *filter*)
Output: a new ontology (Onto *RTO*)

```
/* The first phase */
List < Align > Alignment
  //saving the match results between each pairs in sub_DO
  //Align is 5-tuple in this paper: {e1, e2, r, O1, O2}
  //e.g. {entity 2, entity 8, "≡", sub(DO1), sub(DO2)}
k = 0
for i = 0 to length(sub_DO) − 1:
  for j = i + 1 to length(sub_DO):
    Alignment[k] = ontoMatch(sub_DO[i], sub_DO[j])
      //AML is deployed to match ontologies [36]
    k = k+ 1
  end for
end for
/* The second phase */
/* The function of Onto:
  addEntity(e): add the entity e into ontology
  addRelation(e): add the relationships that e has into
  ontology
  addSuperClass(e): add the entity e into ontology as
  super-class
  addSubClass(e): add the entity e into ontology as
  sub-class
  getSuperClass(e): obtain the super-class of entity e
*/
Onto skele //saving the skeleton of RTO
for i = 0 to length(Alignment):
  swith Alignment.r
    case "≡":
      if filter(Alignment.O1).DoI > filter
      (Alignment.O2).DoI:
        skele.addEntity(Alignment.e1)
      end if
    case "⊆":
      if filter(Alignment.O1).DoI > filter
      (Alignment.O2).DoI:
        skele.addEntity(Alignment.e1)
        Alignment.e1.addSuperClass(Alignment.e2)
        skele.addRelation(Alignment.O1.e1)
      end if
    case "⊥":
      if filter(Alignment.O1).DoI > filter
      (Alignment.O2).DoI:
        skele.addEntity(Alignment.e1)
        Alignment.e1.getSuperClassaddSubClass
        (Alignment.e2)
        skele.addRelation(Alignment.O1.e1)
      end if
  end swith
end for
//Because the elements in sub_DO had been ranked by score of
//themselves, it is no need to deal with the conflictions.
for i = 0 to length(sub_DO):
  for each entity e in sub_DO[i]:
    if e is not in skele:
      skele.addEntity(e)
      skele.addRelation(e)
    end if
  end for
end for
RTO = skele
return RTO
```

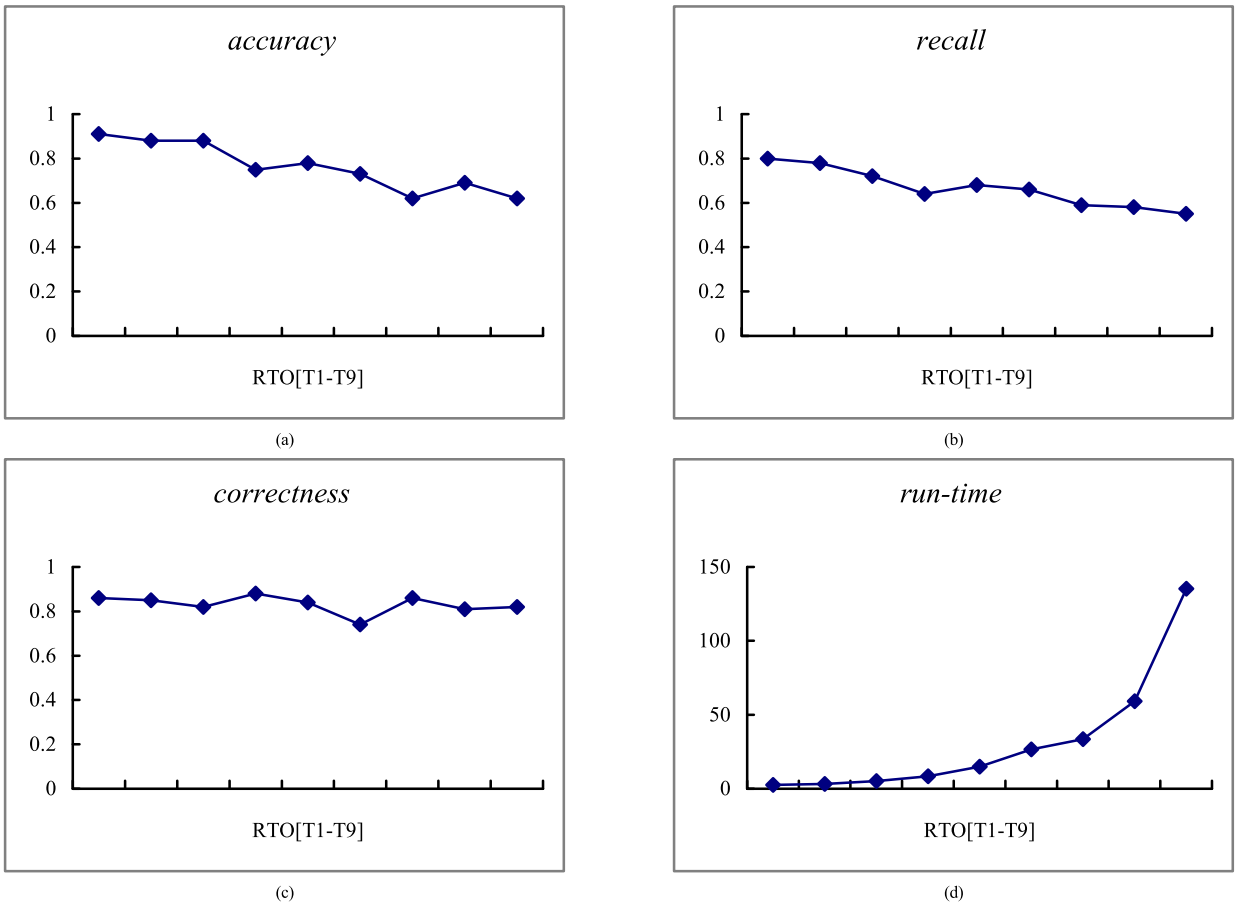


FIGURE 8. Evaluation results of RTO[T_1-T_9]. (The stability of *accuracy*, *recall*, and *correctness* are great; but *run-time* increases exponentially).

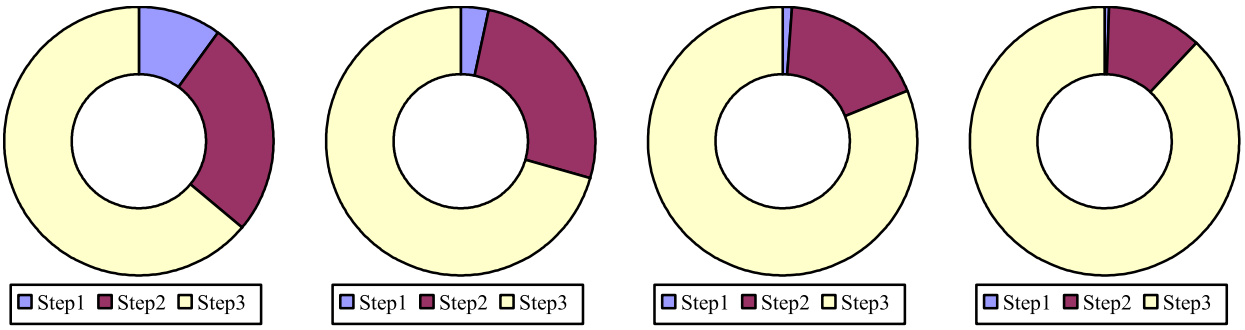


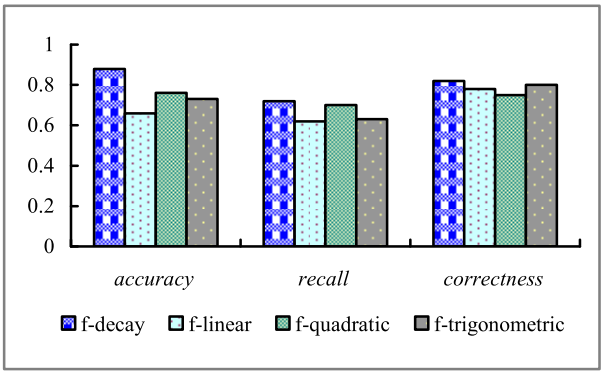
FIGURE 9. The run-time of RTO[T_2], RTO[T_5], RTO[T_8], RTO[T_9]. (As the number of domains increases, Step 3 is the most time-consuming, i.e. the computation complexity of Step 3 is the highest).

effectiveness, the stability of the method was great. As shown in Fig. 8, *run-time* increased exponentially as the value of i increase. Because, while the number of domain ontologies that related to T_i increased, ontology matching was executed more times in Step 3. Even so, the average of *run-time* is low, it is indicated that the efficiency of the method is great also. In addition, RTO is really lightweight. The number of entities in RTO is few very much.

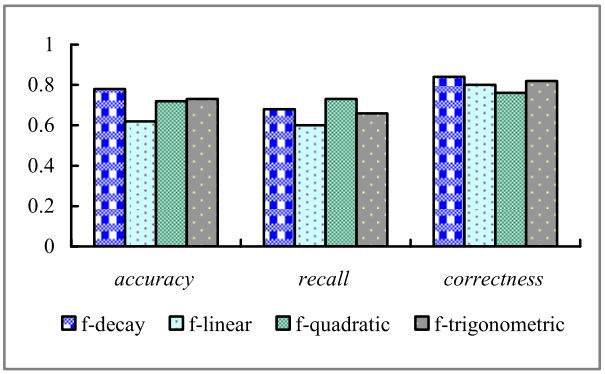
We deployed several excellent methods of ontology matching which were reported by OAEI to integrate all domain ontologies into SLO, such as AML, LogMap, POMap, Lily. SLO[AML] means the SLO built by AML. As shown in Tab. 4, comparing with SLO and TLO, the scale of RTO reduced by 1-2 orders of magnitude almost. A great deal of entities that unrelated to T_i were eliminated in RTO. Notice that, we did not build TLO, because it is a manual process. The data in Tab. 4 about TLO meant the data of the set of all domain ontologies, referring to formula (3). And by the reason for this, we compared *correctness* and *run-time* with only SLO. The average of correctness of RTO

TABLE 4. Evaluation results of RTO, SLO, and TLO.

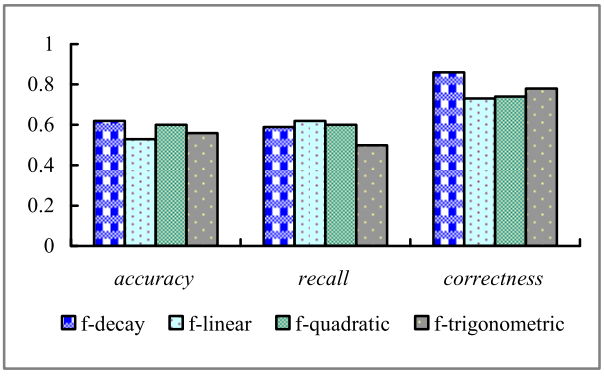
RTO	*Accuracy*	*Recall*	*Correctness*	*Run-time*(s)	Number of concepts	Number of relationships
RTO[T_1]	0.91	0.8	0.86	2.6125	47	151
RTO[T_2]	0.88	0.78	0.85	3.1749	87	223
RTO[T_3]	0.88	0.72	0.82	5.2585	106	294
RTO[T_4]	0.75	0.64	0.88	8.3574	122	397
RTO[T_5]	0.78	0.68	0.84	14.9563	135	395
RTO[T_6]	0.73	0.66	0.74	26.5468	140	391
RTO[T_7]	0.62	0.59	0.86	33.5487	149	452
RTO[T_8]	0.69	0.58	0.81	59.2153	164	486
RTO[T_9]	0.62	0.55	0.82	135.2458	196	534
SLO[AML]	-	-	0.76	854.1025	7615	24963
SLO[LogMap]	-	-	0.68	569.2548	7529	21647
SLO[POMap]	-	-	0.43	2156.2974	7758	24610
SLO[Lily]	-	-	0.51	2685.2541	7954	26509
TLO	-	-	-	-	>10013	>34629



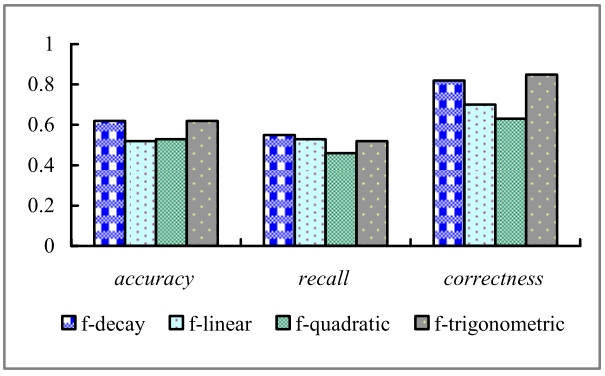
(a) Results with T_3 input



(b) Results with T_5 input



(c) Results with T_7 input



(d) Results with T_9 input

FIGURE 10. Results of *accuracy*, *recall*, and *correctness* among different filters. (*f*-decay introduced in this paper is little greater than others).

was higher than which of SLO. In fact, the lager the scale of ontologies is, the lower the correctness of integration result is.

We chose RTO[T_2], RTO[T_5], RTO[T_8], and RTO[T_9] to tested the *run-time* of each step in the method. The results were illustrated in Fig. 9. The time-consumption of Step 3

was the highest, because ontology matching was a time-consuming process in Step 3. Yet for RTO, the scales of sub-ontologies to be matched were much smaller in Step 3, which is the key reason why building RTO took less time.

In addition, we also test the f-decay of the filter in this paper comparing with other decay functions, such as *linear decay function*, *quadratic decay function*, and *trigonometric decay function*. They are represented as formula (19), (20), and (21), respectively. As shown in Fig. 10, f-decay in this paper is little better than others.

$$f\text{-linear}_i(k) = 1 - (1 - \mathrm{DoI}_i)k \tag{19}$$

$$f\text{-quadratic}_i(k) = 1 - ((1 - \mathrm{DoI}_i)k)^2 \tag{20}$$

$$f - \text{trigonometric}_i(k) = 1 - \sin((1 - \mathrm{DoI}_i)k) \tag{21}$$

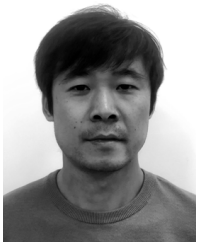
VI. CONCLUSION

To resolve the problem of correlating ontologies for multi-domain knowledge organization, this paper presented a novel ontology model namely RTO. Different from SLO and TLO, RTO took the user requirements as inputs; it only retained the entities that related to user requirements. The method of building RTO based on adaptive filter is presented in this paper. It includes three steps. The details of each step were illustrated in the paper. The evaluation criteria of RTO were introduced, and the experiment was conducted to evaluate RTO. Comparing with SLO and TLO, the correctness of RTO was higher and the time-consumption of RTO was lower. This paper had built 10 domain ontologies and designed 9 user requirements as inputs to evaluate RTO built by each input. With the increasing of the number of relevant domain, it was demonstrated that the accuracy and recall of RTO decreased; the correctness of RTO was stable; and the run-time of RTO increased. In addition, we compared 4 kinds of relevancy decay functions in filter, and the experiment results demonstrated that the exponential decay function used in this paper is little better than others. In future, it will improve the RTO that how to recognize user requirements intelligently and reduce the run-time of ontology matching.

REFERENCES

[1] G. Sirin, E. Coatanéa, B. Yannou, and E. Landel, "Creating a domain ontology to support the numerical models exchange between suppliers and users in a complex system design," in *Proc. Int. Design Eng. Tech. Conf. Comput. Inf. Eng. Conf.*, Feb. 2014, pp. 1–13.
[2] T. Langner and M. Krengel, "The mere categorization effect for complex products: The moderating role of expertise and affect," *J. Bus. Res.*, vol. 66, no. 7, pp. 924–932, Jul. 2013.
[3] E. Lee, B. Ko, C. Choi, and P. Kim, "A design of sensor data ontology for a large scale crop growth environment system," *Scalable Information Systems* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 139. Berlin, Germany: Springer-Verlag, 2014, pp. 91–96.
[4] J. M. Ruiz-Martínez, R. Valencia-García, R. Martínez-Béjar, and A. Hoffmann, "BioOntoVerb: A top level ontology based framework to populate biomedical ontologies from texts," *Knowl.-Based Syst.*, vol. 36, pp. 68–80, Dec. 2012.
[5] L. Ding, R. Pan, and T. Finin, "Finding and ranking knowledge on the semantic Web," in *Proc. 4th Int. Semantic Web Conf.* Berlin, Germany: Springer, 2005, pp. 156–170.
[6] P. Qin, W. Xu, and J. Guo, "A novel negative sampling based on TFIDF for learning word representation," *Neurocomputing*, vol. 177, pp. 257–265, Feb. 2016.
[7] C. Patel, K. Supekar, Y. Lee, and E. K. Park, "OntoKhoj: A semantic Web portal for ontology searching, ranking and classification," in *Proc. 5th ACM Int. Workshop Web Inf. Data Manage. (WIDM)*, Jul. 2003, pp. 1–8.
[8] S. Zhao, K. Prenger, L. Smith, T. Messina, H. Fan, E. Jaeger, and S. Stephens, "Rainbow: A tool for large-scale whole-genome sequencing data analysis using cloud computing," *BMC Genomics*, vol. 14, pp. 1–11, Dec. 2013.
[9] Y. Zhang, V. Wamberto, and S. Derek, "Ontosearch: An ontology search engine," in *Proc. Int. Conf. Innov. Techn. Appl. Artif. Intell.* London, U.K.: Springer, 2004, pp. 58–70.
[10] S. K. Rajapaksha and N. Kodagoda, "Internal structure and semantic Web link structure based ontology ranking," in *Proc. 4th Int. Conf. Inf. Autom. Sustainability*, Dec. 2008, pp. 86–90.
[11] S. Gilani, C. Quinn, and J. J. McArthur, "A review of ontologies within the domain of smart and ongoing commissioning," *Building Environ.*, vol. 182, Sep. 2020, Art. no. 107099.
[12] A. Anikin, D. Litovkin, M. Kultsova, and E. Sarkisova, "Ontology visualization: Approaches and software tools for visual representation of large ontologies in learning," in *Proc. Conf. Creativity Intell. Technol. Data Sci.* Cham, Switzerland: Springer, Aug. 2017, pp. 133–149.
[13] J. Z. Pan, E. Thomas, and D. Sleeman, "Ontosearch2: Searching and querying Web ontologies," in *Proc. IADIS Int. Conf.*, Jan. 2006, pp. 211–218.
[14] B. Fazzinga, G. Gianforme, G. Gottlob, and T. Lukasiewicz, "Semantic Web search based on ontological conjunctive queries," *J. Web Semantics*, vol. 9, no. 4, pp. 453–473, Dec. 2011.
[15] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, and G. Vetere, "DL-Lite: Practical reasoning for rich DLs," in *Proc. Int. Workshop Description Logics*, Jun. 2004, pp. 1–8.
[16] J. Pan, E. Thomas, Y. Ren, and S. Taylor, "Exploiting tractable fuzzy and crisp reasoning in ontology applications," *IEEE Comput. Intell. Mag.*, vol. 7, no. 2, pp. 45–53, May 2012.
[17] F. Bobillo, M. Delgado, and J. Gómez-Romero, "Reasoning in fuzzy OWL 2 with DeLorean," in *Uncertainty Reasoning for the Semantic Web*, 2nd ed. Berlin, Germany: Springer, 2010, pp. 119–138.
[18] U. Straccia, "Answering vague queries in fuzzy DL-Lite," in *Proc. 11th Int. Conf. Inf. Process. Manage. Uncertainty Knowl.-Based Syst.*, EDK, Paris, 2006, pp. 2238–2245.
[19] R. Ungrangsi, C. Anutariya, and V. Wuwongse, "SQORE: An ontology retrieval framework for the next generation Web," *Concurrency Comput., Pract. Exper.*, vol. 21, no. 5, pp. 651–671, Apr. 2009.
[20] R. Ungrangsi, C. Anutariya, and V. Wuwongse, "SQORE-based ontology retrieval system," in *Proc. Int. Conf. Database Expert Syst. Appl.* Berlin, Germany: Springer, Sep. 2007, pp. 720–729.
[21] M. Gao, C. Liu, and F. Chen, "An ontology search engine based on semantic analysis," in *Proc. 3rd Int. Conf. Inf. Technol. Appl. (ICITA)*, Sydney, NSW, Australia, Jul. 2005, pp. 256–259.
[22] R. Ungrangsi, C. Anutariya, and V. Wuwongse, "Enabling efficient knowledge reuse in the semantic Web with SQORE," in *Proc. 3rd Int. Conf. Semantics, Knowl. Grid (SKG)*, Xi'an, China, Oct. 2007, pp. 92–97.
[23] T. Uchibayashi, B. O. Apduhan, and N. Shiratori, "Experiments and functional analysis in integrating sub-ontology extraction and tailoring," in *Proc. Int. Conf. Comput. Sci. Appl.*, Jun. 2011, pp. 143–149.
[24] T. Uchibayashi, B. O. Apduhan, and N. Shiratori, "A domain specific sub-ontology derivation end-user tool for the semantic grid," *Telecommun. Syst.*, vol. 55, pp. 1–11, Jan. 2013.
[25] A. Flahive, D. Taniar, W. Rahayu, and B. O. Apduhan, "Ontology tailoring in the semantic grid," *Comput. Standards Interfaces*, vol. 31, no. 5, pp. 870–885, Sep. 2009.
[26] M. Bhatt, A. Flahive, and C. Wouters, "A distributed approach to sub-ontology extraction," in *Advanced Information Networking and Applications*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2004, pp. 636–641.
[27] T. Uchibayashi, B. O. Apduhan, and N. Shiratori, "A domain specific sub-ontology derivation end-user tool for the semantic grid," *Telecommun. Syst.*, vol. 55, no. 1, pp. 125–135, Jan. 2014.
[28] A. Flahive, D. Taniar, and W. Rahayu, "Ontology as a service (OaaS): Extracting and replacing sub-ontologies on the cloud," *Cluster Comput.*, vol. 16, no. 4, pp. 947–960, Dec. 2013.
[29] M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, and D. Taniar, "MOVE: A distributed framework for materialized ontology view extraction," *Algorithmica*, vol. 45, no. 3, pp. 457–481, Jul. 2006.
[30] A. Flahive, D. Taniar, W. Rahayu, and B. O. Apduhan, "A methodology for ontology update in the semantic grid environment," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 4, pp. 782–808, Mar. 2015.

[31] A. Flahive, D. Taniar, and W. Rahayu, "Ontology as a service (OaaS): A case for sub-ontology merging on the cloud," *J. Supercomput.*, vol. 65, no. 1, pp. 185–216, Jul. 2013.
[32] A. Flahive, D. Taniar, W. Rahayu, and B. O. Apduhan, "Ontology expansion: Appending with extracted sub-ontology," *Log. J. IGPL*, vol. 19, no. 5, pp. 618–647, Oct. 2011.
[33] A. Flahive, J. W. Rahayu, D. Taniar, and B. O. Apduhan, "A distributed ontology framework in the semantic grid environment," in *Advanced Information Networking and Applications*, vol. 2. New York, NY, USA: IEEE Press, 2005, pp. 193–196.
[34] Y. Li, J. Zhou, J. Liu, and Y. Hou, "Matching large scale ontologies based on filter-and-verification," *Math. Problems Eng.*, vol. 2020, pp. 1–8. May 2020.
[35] J. Euzenat and P. Shvaiko, "Part II: Ontology matching techniques," in *Ontology Matching*, 2nd ed. Berlin, Germany: Springer, 2013, pp. 73–84.
[36] I. F. Cruz, F. P. Antonelli, and C. Stroe, "AgreementMaker: Efficient matching for large real-world schemas and ontologies," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1586–1589, Aug. 2009.
[37] D. Faria, C. Pesquita, E. Santos, I. F. Cruz, and F. M. Couto, "AgreementMakerLight 2.0: Towards efficient large-scale ontology matching," in *Proc. Int. Semantic Web Conf. (Posters Demos)*, Oct. 2014, pp. 457–460.
[38] J. Da Silva, K. Revoredo, F. Baião, and J. Euzenat, "Alin: Improving interactive ontology matching by interactively revising mapping suggestions," *Knowl. Eng. Rev.*, vol. 35, pp. 1–22, 2020.
[39] E. Jimenez-Ruiz and B. C. Grau, "LogMap: Logic-based and scalable ontology matching," in *Proc. Int. Semantic Web Conf. (ISWC)*, Oct. 2011, pp. 273–288.
[40] E. Jimenez-Ruiz, B. C. Grau, Y. Zhou, and I. Horrocks, "Large-scale interactive ontology matching: Algorithms and implementation," in *Proc. Eur. Conf. Artif. Intell. (ECAI)*, vol. 242, 2012, pp. 444–449.
[41] M. A. Khoudja, M. Fareh, and H. Bouarfa, "Ontology matching using neural networks: Evaluation for OAEI tracks," in *Proc. Int. Symp. Modelling Implement. Complex Syst.* Cham, Switzerland: Springer, Oct. 2020, pp. 262–276.
[42] P. Wang, "Lily-LOM: An efficient system for matching large ontologies with non-partitioned method," in *Proc. CEUR Workshop*, vol. 658, Nov. 2010, pp. 69–72.
[43] H. Li, A. N. P. Mina, Y. Li, and L. Patrick. (2020). *Results for OAEI 2020—Anatomy Track*. Athens. [Online]. Available: https://oaei.ontologymatching.org/2020/results/anatomy/index.html



JIANHUI ZHOU received the B.S. and M.S. degrees in mechanical engineering and automation and the Ph.D. degree in aerospace manufacturing engineering from Beihang University, China, in 2008, 2012, and 2018, respectively.

He is currently a Lecturer with the School of Computer and Network Engineering, Shanxi Datong University. His research interests include ontology matching, data mining, and knowledge engineering technology.

Dr. Zhou is a member of the China Computer Federation (CCF). He was a recipient of the *Journal of Computer-Aided Design and Computer Graphics* Excellent Student Paper Award, in 2016.

XIAOXIA SONG received the B.S. degree in computer science from Yanshan University, China, in 1998, the M.S. degree in computer software and theory from Guangxi Normal University, China, in 2005, and the Ph.D. degree in intelligent information processing from Xidian University, China, in 2013.

She is currently a Professor with the School of Computer and Network Engineering, Shanxi Datong University. Her current research interests include data mining, compressed sensing theory, and optimization calculation. She is a Senior Member of the China Computer Federation (CCF).

YONG LI received the B.S. degree in computer science from Yanshan University, China, in 1998, the M.S. degree in computer science from the Beijing Institute of Technology, China, in 2005, and the Ph.D. degree in communication and information system from Xidian University, China, in 2015.

He is currently a Professor with the School of Computer and Network Engineering, Shanxi Datong University. His research interests include wireless networks, information theory, and big data. He is a member of the China Computer Federation (CCF).

YUN GAO received the B.S. degree in computer science education from Shanxi University, China, in 1998, and the M.E. degree in computer application technology from Tianjin University, China, in 2006.

She is currently a Lecturer with the School of Computer and Network Engineering, Shanxi Datong University. Her research interests include deep learning, machine learning, and big data analysis. She is a member of the China Computer Federation (CCF).

XULONG ZHANG received the M.Sc. and Ph.D. degrees from the College of Computer Science, Chongqing University, in 2011 and 2017, respectively.

He is currently a Lecturer with the School of Computer and Network Engineering, Shanxi Datong University, Datong, China. He has published more than eight academic articles in peer-reviewed international journals. His research interests include computer virus propagation dynamics, wireless sensor networks, and big data. He is a member of the China Computer Federation (CCF).