Scale reasoning with fuzzy-$\mathcal{EL}^+$ ontologies based on MapReduce

Zhangquan Zhou[1] and **Guilin Qi**[1] and **Chang Liu**[2] and **Pascal Hitzler**[3] and **Raghava Mutharaju**[3]
[1]Southeast University, China
{quanzz, gqig}@seu.edu.cn
[2]Shanghai Jiao Tong University, China
liuchang@apex.sjtu.edu.cn
[3]Wright State University, United States
{pascal.hitzler, mutharaju.2}@wright.edu

Abstract

Fuzzy extension of Description Logics (DLs) allows the formal representation and handling of fuzzy or vague knowledge. In this paper, we consider the problem of reasoning with fuzzy-$\mathcal{EL}^+$, which is a fuzzy extension of EL+. We first identify the challenges and present revised completion classification rules for fuzzy-$\mathcal{EL}^+$ that can be handled by MapReduce programs. We then propose an algorithm for scale reasoning with fuzzy-$\mathcal{EL}^+$ ontologies using MapReduce. Some preliminary experimental results are provided to show the scalability of our algorithm.

1 Introduction

The Web Ontology Language OWL which is essentially based on the description logics, has been designed as one of the major standards for formal knowledge representation and automated reasoning in Semantic Web. OWL 2 EL based on description logic $\mathcal{EL}^{++}$, a restricted language of OWL 2, stands out for its positive complexity results and the sufficient expressive power for many real ontologies, such as the medical ontology Snomed-CT.

However, description logics are not able to represent fuzzy information, which is available in some applications, such as multimedia and bioinformatics. Fuzzy extension of description logics has been proposed using fuzzy sets and fuzzy logics [Klir and Yuan, 1995] to provide more expressive power. One of the challenging problems of fuzzy description logics is reasoning with large scale fuzzy ontologies. Such ontologies can be extracted from different sources, such as multimedia (see [Dalakleidi *et al.*, 2011]).

Parallel reasoning is an obvious choice to easily achieve the scalability goal. There have been some works covering it. One of the most successful attempts is WebPIE [Urbani *et al.*, 2010], an efficient inference engine for large amount of RDF triples under pD* semantics [ter Horst, 2005] using MapReduce framework. This work is further extended in [Liu *et al.*, 2011] to handle fuzzy knowledge. In [Mutharaju *et al.*, 2010] a parallel classification algorithm using MapReduce is given for classical $\mathcal{EL}^+$. However, this algorithm is not optimized for implementation and cannot handle reasoning in fuzzy ontologies. A concurrent method based on multi-core system is discussed in [Kazakov *et al.*, 2011] to reason with classical $\mathcal{EL}$ ontologies, which makes use of multiple cores and the implemented system, ELK performs well in reasoning with large ontologies. A parallel reasoner for $\mathcal{ALC}$ is introduced in [Wu and Haarslev, 2012], which is a tableau-based description logic reasoner.

In this paper, we consider a fuzzy extension of $\mathcal{EL}^+$, called fuzzy-$\mathcal{EL}^+$, which is introduced in [Stoilos *et al.*, 2008]. Although a polynomial time algorithm is given to classify fuzzy-$\mathcal{EL}^+$ ontologies, no experimental evaluation is reported in that work. In order to provide scalable reasoning in fuzzy-EL+, we consider using MapReduce. We first identify the difficulties and challenges to do fuzzy-$\mathcal{EL}^+$ classification using MapReduce framework. We then revise the completion fuzzy-$\mathcal{EL}^+$ rules that can be handled by MapReduce programs and implement a prototype system. We provide some experimental evaluations and prove that our algorithm can scale to fuzzy-EL+ ontologies.

2 Preliminaries

2.1 fuzzy-$\mathcal{EL}^+$

fuzzy-$\mathcal{EL}^+$ is a fuzzy extension of the description logic $\mathcal{EL}^+$, which is introduced in [Stoilos *et al.*, 2008]. Concepts in fuzzy-$\mathcal{EL}^+$ are defined according to the following grammar:

$$C, D ::= \top | A | C \sqcap D | \exists r.C$$

where A ranges over the set of *concept names* (CN) and r over the set of *role names* (RN). A fuzzy-$\mathcal{EL}^+$ ontology is a finite set of *fuzzy general concept inclusions* (F-GCIs) of the form $\langle C \sqsubseteq D, n\rangle$, where $n \in (0, 1]$, and *role inclusions* (RIs) of the form $r_1\circ, ..., \circ r_k \sqsubseteq s$, where k is a positive integer. Note that the *role inclusions* axioms are not fuzzified in [Stoilos *et al.*, 2008].

A polynomial algorithm is given to perform classification of fuzzy-$\mathcal{EL}^+$ ontologies, i.e., it computes all fuzzy subsumptions between concepts of the input ontology $\mathcal{O}$. The algorithm first transforms the given ontology $\mathcal{O}$ into normal form, where all concept inclusions are one of the forms:

$$\langle A_1 \sqcap ... \sqcap A_k \sqsubseteq B, n\rangle$$

$$\langle A \sqsubseteq \exists r.B, n\rangle$$

$$\langle \exists r.B \sqsubseteq A, n\rangle$$

Table 1: **A simple medical ontology**

α_1	ElbowJoint	$\sqsubseteq$	Joint	0.9
α_2	ElbowJoint	$\sqsubseteq$	$\exists$hasLocation.Elbow	0.8
α_3	Joint	$\sqsubseteq$	$\exists$isPartOf.Body	0.6
α_4	$\exists$isPartOf.Elbow	$\sqsubseteq$	PartOfArm	0.8
α_5	hasLocation	$\sqsubseteq$	isPartOf	

Table 2: **A fragment of reasoning**

(1)$\langle$ElbowJoint, 1$\rangle \in S$(ElbowJoint)	
(2)$\langle$Elbow, 1$\rangle \in S$(Elbow)	
(3)$\langle$Joint, 0.9$\rangle \in S$(ElbowJoint)	R1:α_1,(1)
(4)$\langle$ElbowJoint,Body,0.6$\rangle \in R$(isPartOf)	R2:α_3,(3)
(5)$\langle$ElbowJoint,Elbow,0.8$\rangle \in R$(hasLocation)	R2:α_2,(1)
(6)$\langle$ElbowJoint,Elbow,0.8$\rangle \in R$(isPartOf)	R4:α_5,(5)
(7)$\langle$PartOfArm,0.8$\rangle \in S$(ElbowJoint)	R3:α_4,(2)(6)

Table 3: **Completion rules for fuzzy-$\mathcal{EL}^+$**

R1	If $\langle A_1, n_1\rangle \in S(X), ..., \langle A_l, n_l\rangle \in S(X)$, $\langle A_1 \sqcap ... \sqcap A_l \sqsubseteq B, k\rangle \in \mathcal{O}$ and $\langle B, m\rangle \notin S(X)$, where $m = \min\ (n_1, ..., n_l, k)$ then $S(X) := S(X) \cup \langle B, m\rangle$, where $m = \min\ (n_1, ..., n_l, k)$
R2	If $\langle A, n\rangle \in S(X)$, $\langle A \sqsubseteq \exists r.B, k\rangle \in \mathcal{O}$, and $\langle X, B, m\rangle \notin R(r)$, where $m = \min\ (n, k)$ then $R(r) := R(r) \cup \langle X, B, m\rangle$, where $m = \min\ (n, k)$
R3	If $\langle X, Y, n_1\rangle \in R(r)$, $\langle A, n_2\rangle \in S(Y)$, $\langle \exists r.A \sqsubseteq B, n_3\rangle \in \mathcal{O}$, and $\langle B, m\rangle \notin S(X)$, where $m = \min\ (n_1, n_2, n_3)$ then $S(X) := S(X) \cup \langle B, m\rangle$, where $m = \min\ (n_1, n_2, n_3)$
R4	If $\langle X, Yn\rangle \in R(r), r \sqsubseteq s \in \mathcal{O}$, and $\langle X, Yn\rangle \notin R(s)$ then $R(s) := R(s) \cup \langle X, Y, n\rangle$
R5	If $\langle X, Y, n_1\rangle \in R(r)$, $\langle Y, Z, n_2\rangle \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$, and $\langle X, Z, m\rangle \notin R(t)$, where $m = \min\ (n_1, n_2)$ then $R(t) := R(t) \cup \langle X, Z, m\rangle$, where $m = \min\ (n_1, n_2)$

and all role inclusions are of the form $r_1 \circ r_2 \sqsubseteq s$ or $r \sqsubseteq s$. The normalization can be done in liner time [Stoilos *et al.*, 2008]. In the following, we assume that an input ontology $\mathcal{O}$ is in normal form.

The algorithm is formulated by two mappings S and R, where S ranges over subsets of CN×[0, 1] and R over subsets of CN×CN×[0, 1]. Intuitively, $\langle B, n\rangle \in S(A)$ implies $\langle A \sqsubseteq B, n\rangle$ and $\langle A, B, n\rangle \in R(r)$ implies $\langle A \sqsubseteq \exists r.B, n\rangle$. $S(A)$ and $R(r)$ are initialized as follows:

$S(A) = \{\langle A, 1\rangle, \langle \top, 1\rangle\}$, for each class name A in the input ontology $\mathcal{O}$.

$R(r) = \emptyset$, for each role name in $\mathcal{O}$.

Then the two sets $S(A)$ and $R(r)$ are extended by applying the completion rules in Table 3 until no more rules can be applied.

This algorithm runs in polynomial time and it is sound and complete [Stoilos *et al.*, 2008], i.e., after termination on the given ontology $\mathcal{O}$, $\langle A \sqsubseteq B, n\rangle$ if and only if $\langle B, m\rangle \in S(A)$ holds, where $n, m \in$[0,1] and $m \geq n$.

2.2 An example of fuzzy-$\mathcal{EL}^+$ reasoning

We use an example to illustrate the procedure of fuzzy-$\mathcal{EL}^+$ reasoning. A simple medical ontology is given in Table 1 and it is already in normalized form. α_1-α_4 are F-GCI axioms. They express that an elbow joint is a joint (α_1) and has location in elbow (α_2), a joint is a part of body (α_3), a stuff which is a part of elbow is also a part of arm (α_4). Each F-GCI axiom has a fuzzy value. The last axiom (α_5) is a RI axiom which means has-location is more specific than is-part-of.

In Table 2 we show how the consequence subsumptions $\langle$ ElbowJoint$\sqsubseteq$PartOfArm , 0.8 $\rangle$ (corresponding to (7)) and $\langle$ ElbowJoint$\sqsubseteq$ $\exists$isPartOf.Body, 0.6$\rangle$ (corresponding to (4)) can be derived using the reasoning rules in Table 3.

2.3 MapReduce

MapReduce is a programming model for parallel processing over huge data sets [Dean and Ghemawat, 2004]. A MapReduce task consists of two main phases: map phase and reduce phase. Several tasks complete a MapReduce job which solves a specific problem.

In map phase, a user-defined map function receives a key/value pair and outputs a set of key/value pairs. All the pairs sharing the same key are grouped and passed to reduce phase. Then a user-defined reduce function is set up to process the grouped pairs. The outcome of reduce nodes may be the results of the overall job or the intermediate input of following tasks. The grouping procedure between map and reduce phase is called *shuffle* which is the key factor to determine the efficiency of a task. The functionalities of map and reduce nodes can be formulated as:

Map: (key1, value1) $\mapsto$ list(key2, value2),

Reduce: (key2, list(value2)) $\mapsto$ list(value3).

We give an example on how to use MapReduce programs to apply a rule. We use the previous example and consider the rule R2 in Table 3 on α_3 and (3). In a naive reasoning program, the map function first scans the dataset. When the axiom $\langle$Joint$\sqsubseteq$ $\exists$isPartOf.Body, 0.6$\rangle$ is scanned, it generates a key/value pair (key='Joint', value={Joint$\sqsubseteq$ $\exists$isPartOf.Body, 0.6}). When $\langle$Joint, 0.9$\rangle \in S$(ElbowJoint) is scaned, it generates a pair (key='Joint', value ={$\langle$Joint, 0.9$\rangle \in S$(ElbowJoint)}). Then the reduce function processes outputs of map function that share the same key. It iterates the collected values and computes the fuzzy value. Finally the reduce function generates the consequence subsumption $\langle$ ElbowJoint$\sqsubseteq$ $\exists$isPartOf.Body, 0.6$\rangle$.

Here are some principles for designing an efficient MapReduce program:

i Always keep in mind to make full use of the throughout capacity of clusters.

ii Do not increase the burden of shuffle phase unless necessary.

Table 4: **The correspondence between tables and sets**

$(l, A_1, ..., A_l, B, n) \in \mathcal{H}$	$\leftrightarrow$	$\langle A_1 \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle \in \mathcal{O}$,
$(A, B, n) \in \mathcal{I}$	$\leftrightarrow$	$\langle A \sqsubseteq B, n\rangle \in \mathcal{O}$,
$(A, r, B, n) \in \mathcal{J}$	$\leftrightarrow$	$\langle A \sqsubseteq \exists r.B, n\rangle \in \mathcal{O}$,
$(r, A, B, n) \in \mathcal{K}$	$\leftrightarrow$	$\langle \exists r.A \sqsubseteq B, n\rangle \in \mathcal{O}$,
$(r, s) \in \mathcal{L}$	$\leftrightarrow$	$r \sqsubseteq s \in \mathcal{O}$,
$(r, s, t) \in \mathcal{M}$	$\leftrightarrow$	$r \circ s \sqsubseteq t \in \mathcal{O}$,
$(X, A, n) \in \mathcal{S}$	$\leftrightarrow$	$\langle A, n\rangle \in S(X)$,
$(r, X, Y, n) \in \mathcal{R}$	$\leftrightarrow$	$\langle X, Y, n\rangle \in R(r)$.

iii Minimize the number of MapReduce tasks due to the overhead of one MapReduce task.

iv Carefully design the data structures for faster and more efficient processing of MapReduce cluster.

The tradeoff between these principles lead us to design and optimize our algorithms in following work.

3 MapReduce Algorithms for fuzzy-$\mathcal{EL}^+$ Classification

3.1 Challenges in fuzzy-$\mathcal{EL}^+$ reasoning

We summarize two main challenges of reasoning on fuzzy-$\mathcal{EL}^+$ using MapReduce, and give our methods to deal with them.

Translating rules using MapReduce language. The rules in Table 3 can not be directly realized by MapReduce, since the input data for MapReduce should be translated into key/value pairs. We consider to reconstruct the rules in tabular form. In detail, we create eight tables (see Table 4) corresponding to the different forms of axioms in sets $\mathcal{O}, S$ and R, where $\mathcal{H}, \mathcal{I}, \mathcal{J}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{S}, \mathcal{R}$ are tables which contain several tuples, for example the table $\mathcal{I}$ contains the tuples of the form (A, B, n) and these tuples can be easily translated to a key/value pair like (key=$\mathcal{I}$, value = $\{A, B, n\}$). Therefore the rules can be rewritten using the operations on tables. For example, rule R2 is rewritten as:

R2 If $(X, A, n_1) \in \mathcal{S}$, $(A, r, B, n_2) \in \mathcal{J}$ and $(r, X, B, m) \notin \mathcal{R}$, where $m = \min(n_1, n_2)$, then $\mathcal{R} := \mathcal{R} \cup (r, X, B, m)$, where $m = \min(n_1, n_2)$.

The other rules can be rewritten similarly. In this way, the procedure of applying rules can be regarded as the sequences of operations on these tables. For R2, an application of it can be treated as a 2-way join between $\mathcal{S}$ and $\mathcal{J}$ (formulated as $\mathcal{S} \bowtie \mathcal{J}$) in which A is the joint. It is easy to prove the correctness of the rules in tabular form by the correspondence between the eight tables and the axioms in the sets $\mathcal{O}, S$ and R.

Multiple joints. In Table 3, the rules R1, R3 and R5 have multiple joints (more than one) in their preconditions. As mentioned above, these rules can be seen as multi-way joins (for example R3 can be formulated as a 3-way join $\mathcal{R} \bowtie \mathcal{S} \bowtie \mathcal{K}$). It is easy for MapReduce to handle 2-way joins like the example in section 2.3 but not for multi-way joins. In our case, R2 and R4 can be directly handled by MapReduce programs and the remaining rules need to be modified. We give revised fuzzy-$\mathcal{EL}^+$ rules in Table 5 and in following sections, we will discuss why we adopt these modifications.

Table 5: **Revised fuzzy-$\mathcal{EL}^+$ rules**

Key	Completion Rule For MapReduce	
A_j	**R1-1**	If $\langle A_j, n_1\rangle \in S(X)$ and $\langle i, B, l, n_2\rangle \in T(A_j)$ then $P(X) := P(X) \cup \{\langle i, A_j, B, l, m\rangle\}$, where $m = \min(n_1, n_2)$, $i = (\langle A_1 \sqcap ... \sqcap A_j \sqcap ... \sqcap A_l \sqsubseteq B, n_2\rangle)$
$A.i$	**R1-2**	If $\langle i, A_1, B, l, n_1\rangle \in P(X)$,..., $\langle i, A_{l-1}, B, l, n_{l-1}\rangle \in P(X)$ and$\langle i, A_l, B, l, n_l\rangle \in P(X)$ then $S(X) := S(X) \cup \{\langle B, m\rangle\}$, where $m = \min(n_1, ..., n_{l-1}, n_l)$
A	**R2**	If $\langle A, n_1\rangle \in S(X)$ and $\langle A \sqsubseteq \exists r.B, n_2\rangle \in \mathcal{O}$ then $R(r) := R(r) \cup \{\langle X, B, m\rangle\}$, where $m = \min(n_1, n_2)$
A	**R3-1**	If $\langle A, n_1\rangle \in S(Y)$ and $\langle \exists r.A \sqsubseteq B, n_2\rangle \in \mathcal{O}$ then $Q(X) := Q(r) \cup \{\langle Y, B, m\rangle\}$, where $m = \min(n_1, n_2)$
$Y.r$	**R3-2**	If $\langle X, Y, n_1\rangle \in R(r)$ and $\langle Y, B, n_2\rangle \in Q(r)$ then $S(X) := S(X) \cup \{\langle B, m\rangle\}$, where $m = \min(n_1, n_2)$
r	**R4**	If $\langle X, Y, n\rangle \in R(r)$ and $r \sqsubseteq s \in \mathcal{O}$ then $R(s) := R(s) \cup \{\langle X, Y, n\rangle\}$
Z	**R5**	If $\langle X, Z, n_1\rangle \in R(r)$ and $\langle Z, Y, n_2\rangle \in R(s)$ and $r \circ s \sqsubseteq t \in \mathcal{O}$ then $R(t) := R(t) \cup \{\langle X, Y, m\rangle\}$, where $m = \min(n_1, n_2)$

3.2 Handling R1

The rule R1 handles the *concept conjunction inclusion* (CCI) axioms. A CCI axiom is in the form of $\langle A_1 \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle$, where $l \geq 1$ ($\langle A \sqsubseteq B, n\rangle$ is a special case with $l = 1$). The application of R1 is a complex multi-way join on tables $\mathcal{S}$ and $\mathcal{H}$,and it is not intuitive to split the multi-way join into several 2-way joins.

In order to handle R1 using MapReduce programs, we first introduce a function I and a mapping T. The function I assigns each CCI axiom in input ontology $\mathcal{O}$ an integer that is used as the identifier of the axiom. To create T, we first set $T(X)$ as $\emptyset$ for each X in CN. Then for every CCI axiom like $\langle A_1 \sqcap ... \sqcap A_j \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle$, each concept A_j $(i \leq j \leq l)$ occurring in the left side of B is checked and its corresponding set $T(A_j)$ is extended as:

$T(A_j) := T(A_j) \cup \{\langle i, B, l, n\rangle\}$ where $i = I(\langle A_1 \sqcap ... \sqcap A_j \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle)$.

In this way, $\langle A_1 \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle$ can be replaced by $\langle i, B, l, n\rangle \in T(A_1)$,..., $\langle i, B, l, n\rangle \in T(A_l)$.

We get T and I before reasoning and use a new mapping P to split R1 into R1-1 and R1-2 in Table 5. $\langle i, A_j, B, l, m\rangle \in P(X)$ ($i = (\langle A_1 \sqcap ... \sqcap A_j \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle)$) means that X is subsumed by A_j with the fuzzy value m and A_j occurs in the concept conjunction of the axiom

Table 6: **Effects of intermediate results**

ontology		$\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}$	$\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{K}}$	$\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$
GALEN	no. of tuples/M	163	1638	40
	utilization ratio	1.61%	1.36%	12.81%
	total cost time of R3/minutes	33	580	27

$\langle A_1 \sqcap ... \sqcap A_j \sqcap ... \sqcap A_l \sqsubseteq B, n\rangle$. $P(X)$ contains the intermediate or incomplete derived information that is used only in R1-2 to complete the work of R1. The length of conjunction l is recorded in $P(X)$ and used in R1-2 to check whether all concepts in conjunction ($A_1 \sqcap ... \sqcap A_l$) are collected.

If we transform the mappings T and P to tables $\mathcal{T}$ and $\mathcal{P}$, the application of R1 can be represented by a sequence of two 2-way joins, respectively $\mathcal{S} \bowtie \mathcal{T}$ and $\mathcal{P} \bowtie \mathcal{P}$, which can be easily handled by MapReduce programs, i.e., we can use two MapReduce tasks to handle R1-1 and R1-2. The keys $A.i$ in the left side of R1-2 means that A and i are both used to construct the key.

3.3 Handling R3

As mentioned in section 3.1, the application of R3 can be seen as a 3-way join on tables $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{K}$, formulated as:

$$\mathcal{R}(r, X, Y, n_1) \bowtie \mathcal{S}(Y, A, n_2) \bowtie \mathcal{K}(r, A, B, n_3)$$

This 3-way join can be completed by two 2-way joins because the two joins in the 3-way join can be done sequentially, i.e., we can first join two of the three tables and then join the intermediate result with the third one. Since any two tables share a common joint, there are three join orders to split the 3-way join into two steps as follows:

$(\mathcal{R}(r, X, Y, n_1) \bowtie \mathcal{S}(Y, A, n_2)) \bowtie \mathcal{K}(r, A, B, n_3)$ $\bowtie_1$
$(\mathcal{R}(r, X, Y, n_1) \bowtie \mathcal{K}(r, A, B, n_3)) \bowtie \mathcal{S}(Y, A, n_2)$ $\bowtie_2$
$(\mathcal{S}(Y, A, n_2) \bowtie \mathcal{K}(r, A, B, n_3)) \bowtie \mathcal{R}(r, X, Y, n_1)$ $\bowtie_3$

The distinctions in performance among those three orders are just decided by the intermediate join results, because in the first hand, the tables $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{K}$ will be read once and joined with other tables in *reduce* function, thus the overheads these three tables contribute to is in equality (formulated as $|\mathcal{R}| + |\mathcal{S}| + |\mathcal{K}|$). In the second hand, the three join orders will output the same results, so the overheads contributed by the final results are also in equality. We use tables $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}$, $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{K}}$ and $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$ to respectively denote the intermediate results of $\bowtie_1$, $\bowtie_2$ and $\bowtie_3$ (namely the results of first joins in brackets). We did an experiment[1] to investigate the overheads contributed by $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}$, $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{K}}$ and $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$. The experimental results on GALEN are listed in Table 6.

In this experiment, we accumulate the number of tuples of $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}$, $\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{K}}$ and $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$ generated in all iterations of reasoning. The total size of the three tables is listed in first row. The second row shows the utilization ratio of the three tables, which denotes the percentage of the tuples used in second joins. The total cost times of R3 are collected in the third row.

From the experiment results, we find that $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$ has the smallest size and the highest utilization ratio, and its corresponding join order costs minimal time compared to the other two orders. We have the same results on other ontologies.

We give a brief analysis here. The size of $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$ is $p|\mathcal{S}||\mathcal{K}|$, where p is the probability of two tuples from $\mathcal{S}$ and $\mathcal{K}$ agreeing on their common joint. Since $\mathcal{S} \bowtie \mathcal{K}$ is on the joint A, namely the named concepts, p can be estimated as $\frac{|CN|}{|CN|^2}$ (= $\frac{1}{|CN|}$). Therefore, we estimate that $|\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}| \approx \frac{|\mathcal{S}|\cdot|\mathcal{K}|}{|CN|}$. $\mathcal{R} \bowtie \mathcal{K}$ is on r (roles), we can estimate that $|\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{J}}| \approx \frac{|\mathcal{R}|\cdot|\mathcal{J}|}{|RN|}$, similarly, $|\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}| \approx \frac{|\mathcal{R}|\cdot|\mathcal{S}|}{|CN|}$. As we see, the table $\mathcal{K}$ keeps unmodified, so its size ($|\mathcal{K}|$) is fixed. In another respect, $\mathcal{S}$ and $\mathcal{J}$ are expanding and generally larger than $\mathcal{K}$ during the reasoning. Thus $|\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{S}}|$ is always bigger than $|\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}|$. We have an observation that the number of roles is much less than that of concepts ($|RN| \ll |CN|$) in most real ontologies, so $|\mathcal{Q}_{\mathcal{R}\bowtie\mathcal{J}}|$ is always the biggest one among the three intermediate results, which is also consistent with the experimental results.

We adopt the join order $\bowtie_3$ and introduce a new mapping Q to split R3 into R3-1 and R3-2 in Table 5. Q records the intermediate result of R3 (corresponding to $\mathcal{Q}_{\mathcal{S}\bowtie\mathcal{K}}$). Intuitively, $\langle Y, B, n\rangle \in Q(r)$ implies $\langle \exists r.A \sqsubseteq B, n\rangle$. $Q(r)$ is initially set to $\emptyset$ for each role r. R3-1 and R3-2 can be handled by MapReduce programs.

3.4 Loading role inclusion axioms into memory

The Application of R5 can be seen as a 3-way join on two tables $\mathcal{R}$ and $\mathcal{M}$ as:

$$\mathcal{R}(r, X, Z, n_1) \bowtie \mathcal{R}(s, Z, Y, n_2) \bowtie \mathcal{M}(r, s, t)$$

This rule is unmodified because we have the observation that the number of role inclusion axioms (resp. roles) of the form $r \sqsubseteq s$ or $r \circ s \sqsubseteq t$ is much less than that of the concept inclusion axioms (resp. concepts) in some real ontologies like Snomed-CT and GALEN.

Therefore we assume the role inclusion axioms fit in memory, so that we parallelize the axioms of the form $\langle X, Y, n\rangle \in R(r)$ into different map nodes and load the axioms of property chain into memory to complete the application of R5. To illustrate the process, we give Algorithm 1 and Algorithm 2 which correspond respectively to *map* function and *reduce* function of R5. For simple understanding, these two algorithms are described using the rule in Table 5.

Algorithm 1 Map function for R5

Input: key, $\langle X, Y, n_1\rangle \in R(r)$ as the value

```
for each $r \circ s \sqsubseteq t \in \mathcal{O}$ do
    emit(key:Y, value:$\langle X, Y, n_1\rangle \in R(r)$)
end for
for $s \circ r \sqsubseteq t \in \mathcal{O}$ do
    emit(key:X, value:$\langle X, Y, n_1\rangle \in R(r)$)
end for
```

The *map* function completes two joins ($\mathcal{R}(r, X, Z, n_1) \bowtie \mathcal{M}(r, s, t)$ and $\mathcal{R}(s, Z, Y, n_2) \bowtie \mathcal{M}(r, s, t)$). The results of the two joins are processed in *reduce* function. Therefore we

[1] This experiment is done in a cluster with 8 nodes, in which each node have 2G memory and runs two units for map or reduce function.

Algorithm 2 Reduce function for R5

```
Input: key, iterator values
for each $\langle X, Z, n_1 \rangle \in R(r)$ in values do
    for each $\langle Z, Y, n_2 \rangle \in R(s)$ in values do
        for each $r \circ s \sqsubseteq t \in \mathcal{O}$ do
            m := $\min(n_1, n_2)$
            emit($\langle X, Y, m \rangle \in R(t)$)
        end for
    end for
end for
```

can use only one MapReduce task to handle R5, which helps reduce the numbers of MapReduce tasks. Inspired by the treatment for R5, we can also further optimize the application of R4. We first compute the *role inclusion closure* (RIC) which stands for the reflexive transitive closure of the axiom $r \sqsubseteq s$ in $\mathcal{O}$. When any new axiom $\langle X, Y, n \rangle \in R(r)$ is obtained, namely after applying R2 and R5, we call Algorithm 3 to do further process with the loaded RIC. We use $r \sqsubseteq^* s$ to describe that role r is semantically subsumed by the role s, which is explicit in RIC.

Algorithm 3 applyRIC for R2 and R5

```
Input: $\langle X, Y, n \rangle \in R(r)$ : the inferences of R2 and R5
for each $s$ in $\mathcal{O}$ do
    if $r \sqsubseteq^* s$ is in RIC then
        emit($\langle X, Y, n \rangle \in R(s)$)
    end if
end for
```

This method allows R4 being omitted from the reasoning iteration, thus there is no need to consider the I/O overheads and *map-out* of R4. Since we choose not to fuzzify role axioms as well as [Stoilos *et al.*, 2008], the application of RIC has no effects on the fuzzy values of other derived axioms. We call the function *applyRIC* in the *reduce* function of R2 and R5 to complete the inference task of R4. The *reduce* function of R2 is given by Algorithm 4 to illustrate how to finish the application of R4.

Algorithm 4 Reduce function for R2

```
Input: key, iterator values
for each $\langle A, n_1 \rangle \in S(X)$ in values do
    for each $\langle A \sqsubseteq \exists r.B, n_2 \rangle \in \mathcal{O}$ in values do
        m := $\min(n_1, n_2)$
        emit($\langle X, B, m \rangle \in R(r)$)
        applyRIC($\langle X, B, m \rangle \in R(r)$)
    end for
end for
```

3.5 Overview of the reasoning algorithm

We first discuss the rationales of these revised rules. Rules R2, R4 and R5 are almost unchanged except the preconditions like $\langle B, m \rangle \notin S(X)$ or $\langle X, B, m \rangle \notin R(r)$ are omitted, as they are only used for termination judgment. Since we will consider the termination condition in our reasoning algorithm, there is no difference between these rules in Table 3 and Table 5. Rule R1 (resp. rule R3) is replaced by R1-1 and R1-2 (resp. R3-1 and R3-2). The outputs of R1-1 (resp. R3-1) are only used in the precondition of R1-2 (resp. R3-2), so it does not have any effect on final results.

We then give the reasoning algorithm based on the revised fuzzy-$\mathcal{EL}^+$ rules.

Before reasoning, we first transform all input axioms to normalized forms and initializes S, R, P and Q. The main part of the reasoning work is given by Algorithm 5, which consists of two phases. The first phase is preprocessing, in which Algorithm 5 creates the mapping T and computes the complete *role inclusion closure* (RIC). The second phase is reasoning, in which Algorithm 5 iteratively applies the fuzzy-$\mathcal{EL}^+$ rules until a fix point is reached. At the end of each iteration, a MapReduce task is used to delete the duplicates and get the greatest fuzzy value for an axiom obtained from completion rules. When there is no new axiom generated, the algorithm terminates.

Algorithm 5 Fuzzy-$\mathcal{EL}^+$ reasoning

```
create the mapping $T$;
RIC := computeRIC();
firstTime := true;
derived = 0;
while firstTime or derived $\geq 0$ do
    derived := applyRules();
    firstTime := false;
end while
```

The application of each rule can be handled by a MapReduce task. In map phase, each axiom which satisfies one of the preconditions of the rule is given as output in form of a key/value pair , where key is concept or role as shown in the left part of Table 5. All axioms having the same key are grouped from different map nodes and passed to one reduce node. The conclusions of the rule can be achieved in reduce phase. Since we can load the axioms of property chain into different nodes, the application of R5 can be done in one MapReduce task. We use RIC in the reduce phases of R2 and R5 to complete the inference task of R4.

4 Experiments

We implemented a prototype system based on a popular implementation of MapReduce model, Hadoop[2], which is an open-source Java implementation project under the Apache Foundation.

Since there is no optimized fuzzy DL system for fuzzy-$\mathcal{EL}^+$, we validate the correctness of our system against jCEL which is a reasoner handling EL ontologies. We run our system on the revised versions of test ontologies, i.e., we manually add fuzzy values to each axiom in these ontologies. Our system can produce the same results as jCEL without considering fuzzy values.

[2] http://hadoop.arpache.org/

Table 7: **Comparison of reasoning time (in seconds)**

Test Datasets	ELK	jCEL	Pellet	Our system (8 nodes)
1-GALEN	2.3	116.2	742.4	6552.5
2-GALENs	5.5	243.7	-	11952.5
4-GALENs	11.6	-	-	19908.3
8-GALENs	-	-	-	38268.7

Table 8: **Scalability over data volume**

Test Datasets	Input (No. of axioms (K))	Output (No. of axioms (K))	Time (hours)	Throughput (Axioms(K) /minutes)
f-1-GALEN	90	6,838	1.82	62.62
f-2-GALENs	178	13,680	3.32	68.67
f-4-GALENs	352	27,349	5.53	82.42
f-8-GALENs	703	54,699	10.63	85.76

The experiments were run in a Hadoop cluster containing 8 nodes. Each node is a PC machine with a 2-core, 3GHz, E8400 CPU, 2GB main-memory and 500G hard disk. In the cluster, each node is assigned two processes to run *map* tasks, and two processes to run *reduce* tasks. So the cluster allows program running on 16 mappers or 16 reducers simultaneously.

4.1 Test datasets

To compare our system with other reasoners and test its scalability, we generate fuzzy-$\mathcal{EL}^+$ ontologies based on GALEN, called f-GALEN, for experimental purpose. In detail, for the normalized GALEN we assign a random fuzzy value f ($0 < f \leq 1$) to each GCI axiom and keep RI axioms unfuzzified.

In order to validate the scalability of our algorithms, we need to run our system on datasets with different sizes to see the relation between the data volume and the throughput. For this purpose, we use a simple method which uses GALEN as a core and generates different number of copies based on the core, and these copies are independent. For the GALEN copies (here n-GALENs denotes to n copies of GALEN) with different sizes, we add fuzzy values to them and get the fuzzy versions (f-n-GALENs).

4.2 Comparison with memory-based reasoners

We compared the reasoning time with three memory-based reasoners ELK, jCEL and Pellet. ELK is a concurrent reasoner using multiple cores [Kazakov *et al.*, 2011]. jCEL is a java implementation of CEL [Baader *et al.*, 2006] for EL reasoning [Mendez, 2012]. Pellet is a tableau-based reasoner for OWL DL [Sirin and Parsia, 2004]. We ran these reasoners in one node of the cluster. In comparison we ran our system in the cluster without considering fuzzy values. The experimental results are given in Table 7. From the results we can see that for memory-based reasoners, the classification will finish when the input datasets fit in memory. For 8-GALENs, none of the three reasoners can finish classification with such memory that is given to them. Our system will finish reasoning on the four datasets using this cluster.

Table 9: **Scalability over number of mappers**

No.of units	Time(hours)	Speedup
16	1.82	2.24
8	2.02	2.02
4	2.80	1.46
2	4.09	1.00

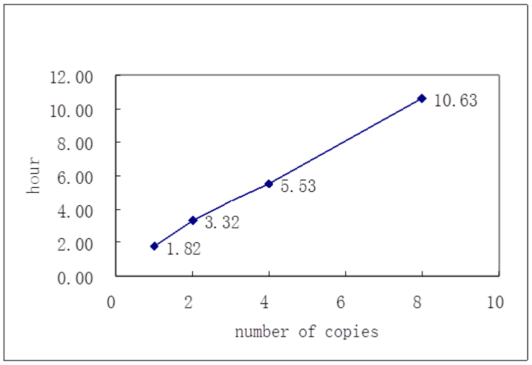

Figure 1: Time versus number of copies

4.3 Scalability tests

To test the scalability of our algorithms, we ran two experiments. The first experiment ran on the cluster with 8 nodes (16 processing units), and handles four datasets with different sizes, they are f-1-GALEN, f-2-GALENs, f-4-GALENs and f-8-GALENs. We give the experimental results in Table 8 to show the relation between the data volume and the throughput. In the second experiment we ran our system on f-1-GALEN with different number of processing units (mappers and reducers) to see the relation between the processing units and the throughput.

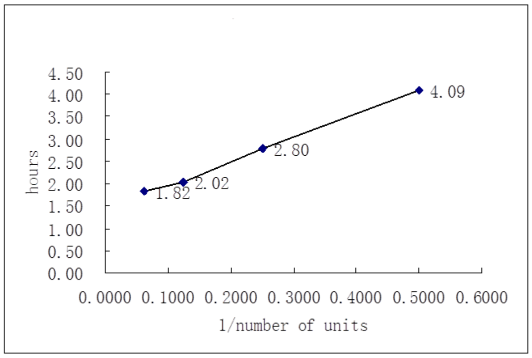

Figure 2: Time versus inverse of number of mappers

From the results of the first experiment, we can see that the throughput increases while the the size of datasets increases. Specially, when the test dataset changes from f-2-GALENs to f-4-GALENs, the throughput increases significantly and the throughput while handling f-8-GALENs is 37% higher than the throughput while handling f-1-GALEN.

Since the cluster has overheads in startup, data transmission and processing, the speedup is non-linear shown in the results of the second experiments (see Table 9). Without considering the overheads and ignoring the constant from the time dimension, we can see that the reasoning time is proportional to the number of copies (see Figure 1) and inversely proportional to the number of units (see Figure 2). For the test datasets, the scalability of our system is validated from the experiments.

5 Conclusion and future work

In this paper, we proposed MapReduce algorithms for classifying ontologies based on fuzzy-$\mathcal{EL}^+$ (it is an extension of $\mathcal{EL}^+$ with fuzzy vagueness). We identified two main challenges using MapReduce for fuzzy-$\mathcal{EL}^+$ reasoning and proposed our solutions for tackling them. We revised the original rules and gave the classification algorithms using MapReduce framework. Furthermore, we implemented a prototype system for the evaluation. The experimental results show that this system has scalability and it can finish the work of classification through adding nodes in the cluster when ontologies do not fit in memory.

The memory-based reasoners mentioned in experiments cannot handle SNOMED-CT in single node because of the memory limit. Our system can process SNOMED-CT, although it cost nearly two days to finish the whole classification in our cluster.

In our next step, we will test the scalability using a larger cluster on the copies of SNOMED-CT and the ontologies in which SNOMED-CT is merged with other medical ontologies. We will also further optimize our algorithm based on following analysis of the limits of our system. 1) Since our system is based on fixed-point algorithms, it will scan the whole rules to check whether there are new axioms generated in each iteration. This costs overheads for reasoning. 2) Our system will set up same nodes for every rule in each iteration, however some rules will do more work than others on special datasets. We can get the statistic information of input ontologies and balance the usage of nodes for rules. 3) Hadoop will rescan all axioms and collect them for application of one rule R in every iteration. However these collected axioms do not need to be rescanned for R. So they can be processed in local node when applying R.

For test data, we would like to use the tool LogMap[3] to get a fuzzy-$\mathcal{EL}^+$ ontology, i.e., we merge two ontologies with the mapping results and use the measures of similarity as fuzzy values. We also consider to extend fuzzy-$\mathcal{EL}^+$ to process ABox datasets [Ren *et al.*, 2011], since ABox datasets are always beyond the memory capacity.

Acknowledgments

Guilin Qi is partially supported by the NSFC grant 61272378. Pascal Hitzler and Raghava Mutharaju are partially supported by the National Science Foundation under award 1017225 III: Small: TROn - Tractable Reasoning with Ontologies.

References

[Baader *et al.*, 2006] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Cel - a polynomial-time reasoner for life science ontologies. In *International Joint Conference on Automated Reasoning*, 2006.

[Dalakleidi *et al.*, 2011] Kalliopi Dalakleidi, Stamatia Dasiopoulou, Giorgos Stoilos, Vassilis Tzouvaras, Giorgos B. Stamou, and Yiannis Kompatsiaris. Semantic Representation of Multimedia Content. In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, 2011.

[Dean and Ghemawat, 2004] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Operating Systems Design and Implementation*, 2004.

[Kazakov *et al.*, 2011] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. Concurrent Classification of EL Ontologies. In *International Semantic Web Conference*, 2011.

[Klir and Yuan, 1995] George J. Klir and Bo Yuan. *Fuzzy sets and fuzzy logic - theory and applications*. Prentice Hall, 1995.

[Liu *et al.*, 2011] Chang Liu, Guilin Qi, Haofen Wang, and Yong Yu. Large Scale Fuzzy pD* Reasoning Using MapReduce. In *International Semantic Web Conference*, 2011.

[Mendez, 2012] Julian Mendez. jcel: A modular rule-based reasoner. In *International Workshop on OWL Reasoner Evaluation (ORE 2012)*, 2012.

[Mutharaju *et al.*, 2010] Raghava Mutharaju, Frederick Maier, and Pascal Hitzler. A Mapreduce Algorithm for $\mathcal{EL}^+$. In *Description Logics*, 2010.

[Ren *et al.*, 2011] Yuan Ren, Jeff Z. Pan, and Kevin Lee. Parallel abox reasoning of el ontologies. In *Joint International Semantic Technology Conference*, 2011.

[Sirin and Parsia, 2004] Evren Sirin and Bijan Parsia. Pellet: An owl dl reasoner. In *Description Logics*, 2004.

[Stoilos *et al.*, 2008] Giorgos Stoilos, Giorgos B. Stamou, and Jeff Z. Pan. Classifying Fuzzy Subsumption in Fuzzy-$\mathcal{EL}^+$. In *Description Logics*, 2008.

[ter Horst, 2005] Herman J. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In *International Semantic Web Conference*, 2005.

[Urbani *et al.*, 2010] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri E. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In *European Semantic Web Conference*, 2010.

[Wu and Haarslev, 2012] Kejia Wu and Volker Haarslev. A parallel reasoner for the description logic alc. In *Description Logics*, 2012.

[3] http://www.cs.ox.ac.uk/isg/projects/LogMap/