基于众包的知识库索引对齐算法

沈秉文 冯建华
(清华大学计算机科学与技术系 北京 100084)

摘 要 知识库对齐工作是近年来的热点研究问题.知识库对齐是将不同知识库中的实体、关系和类型进行对齐.由于知识库的规模巨大,并且不同知识库的结构差异太大,对齐工作有很多问题有待解决.从这些问题出发,且针对知识库里三种属性值,该文提出了三种索引结构,分别是基于字符串的前缀倒排索引、基于日期的DateTrie、基于数字的线段树,并且通过指示函数将对齐的字面值传递到实体对齐,再利用实体与实体之间的结构性提高准确性,最后,采用机器和人工相结合的方式,控制一定的人工预算,减少问题的候选集,利用众包将类别进行对齐,提高准确性.该文在Yago-DBpedia上对比了所提出的方法、PARIS和Exact-string方法.PARIS得到的实体对达到93.5%的准确率和71.2%的召回率,耗时839 min,而Exact-string方法耗时只有1 min,但是召回率只有57.2%.相比于这两种方法,该文的方法达到90%的准确率和79.3%的召回率,耗时25 min,耗时比PARIS方法短,而召回率比Exact-string高.

关键词 知识库对齐;实体对齐;索引;类别对齐;众包
中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2018.01814

Crowdsourcing Knowledge Base Index Alignment

SHEN Bing-Wen FENG Jian-Hua
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Academia and industry have emerged more and more large scale knowledge base and knowledge base alignment work has been a hot research problem in recent years. Because these knowledge bases are from different data sources, and use different technologies to integrate data, will inevitably lead to knowledge in different knowledge base repetitive and complementary. Knowledge base alignment is to align different entities, relationships and the classes of entities. However, due to the enormous size of the knowledge base, and large structural differences in the knowledge base, there are many problems and challenges to be resolved in knowledge alignment work. The existing knowledge base alignment research is mainly based on the similarity function to match or based on the probability model for iterative calculation. The iterative calculation method has high accuracy, but the computation time is long, and the similarity function greedy matching can only match the entities in the knowledge base, and there are some limitations. Starting from these challenges and three kinds of property values, this paper proposes three indexes which is based on string prefix inverted index, date DateTrie, number segment tree, and use the indicative function to pass through the alignment to entities alignment. First, since there are not many relationships and attributes in the knowledge base, we will manually align the relations and attributes of the two knowledge bases. And for each pair of aligned relationships and attributes, we calculate the value of the indicative function. Then for the different property values, set up the above three kinds of index. The results obtained by the three indexes are

收稿日期:2016-12-13;在线出版日期:2017-05-27.**沈秉文**,男,1992年生,硕士,主要研究方向为数据库.E-mail:shenbw0320@gmail.com.
冯建华,男,1969年生,博士,教授,主要研究领域为数据库、结构和半结构数据查询、数据清洗与集成、众包管理.

selected. The similarity is superimposed and the highest degree of similarity is the final set of the correct pair. Third, the alignment of the entity to pass through the relationship to another pair of entities. Treat the aligned entity as the object in the triple, and use predicate's indicative function, calculate the similarity of the subject. We can get all the entities pairs to improve accuracy. Finally, we use of machines and artificial combination to control certain labor budget and reduce the candidate set of classes pairs, and use crowdsourcing to align classes which maximize accuracy. In this paper, we first experimented on the IIMD dataset. Through the experimental data we can see that using the exact same string to align the entity can achieve *F*-measure to be only 40%, time-consuming 1 s, and PARIS with three iterations after convergence, time-consuming 6 seconds, and the proposed method in this paper use 2 seconds to achieve 95.7% *F*-measure. Then we contrast this paper's method, PARIS and Exact-string method on Yago-DBpedia dataset. The two knowledge bases achieved 2219241 pairs of real identical pairs through the identical URIs to get the golden truth. PARIS obtained entity pairs that achieved 93.5% accuracy and a recall rate of 71.2%, which took 839 minutes while the Exact-string method cost time is 1 minute, but the recall rate is only 57.2%. Compared with these two methods, this paper's method achieves 90% accuracy and the recall rate of 79.3%, takes 25 minutes, is shorter than the PARIS method, and the recall rate is higher than that of Exact-string.

Keywords　knowledge base alignment; entities alignment; index; classes alignment; crowdsourcing

1　引　言

近年来，学术界和工业界出现了越来越多的大规模数据量的知识库，例如，Yago[1]、Freebase[2]、Probase[3]、DBpedia[4]等等. 由于这些知识库都是各自从不同的数据源，并且用了不同的技术来集成数据，必然会导致各个知识库之间会有知识重复和互补. 而获取完整的知识，可能会需要集成和复用不同知识库里的知识数据. 知识库对齐可以确定两个不同的知识库中的实体是否指向现实世界中的同一实体，以此来达到知识的集成和清洗的目的. 因此，知识库对齐近年来受到越来越多的关注.

现有的知识库对齐研究主要基于相似性函数进行匹配或者基于概率模型进行迭代计算. 迭代计算的方法虽然准确度高，但是计算时间长，利用相似形函数贪心匹配只能匹配知识库中实体，存在一定的局限性.

知识库对齐研究存在以下几点挑战：首先是复杂度. 知识库拥有大量实体，例如 Freebase 拥有 1300 万实体，Probase 有 1600 万实体. 普通的做法就是遍历所有的实体，因此会有 1600 万×1300 万实体对需要进行比较，复杂度过高. 第二是知识库中有很多除了实体，还有类别，而不同的知识库中的分类树的大小规模存在非常大的差异，如何对齐分类树，并且在一定人工预算下也是一个很复杂的问题.

现有的知识库中的实体的属性值定义域分为：字符串、日期和数字，本文针对这三种属性值定义域以及以上提出的挑战，提出了三种索引结构，分别基于字符串、基于日期和基于数字，通过这些索引结构以及函数定义，大大提高了实体对齐的效率和准确度. 本文也提出采用机器和人工相结合的方式，控制了一定的人工预算，也有相对较高的准确度.

综上所述，本文的贡献在于：

(1) 本文提出了基于字符串、日期、数字的三种索引，加快实体对齐速度，提高效率，并利用了实体的结构性来提高准确性.

(2) 本文提出了基于众包的算法来对齐类别，提高准确性.

(3) 文中提出的方法在真是数据集上进行实验，结果表明，本文方法效率较高，并且有较高准确性.

本文第 2 节对相关工作进行综述；第 3 节给出问题的形式化描述；第 4 节介绍对齐知识库的整个

方法框架;第 5 节提出基于不同字面值的索引结构,并给出基于该索引结构的对齐算法;第 6 节给出利用众包进行类别对齐的算法;第 7 节报告实验的结果;最后,第 8 节对全文进行总结.

2 相关工作

传统的知识库对齐工作主要研究如何对齐类别[5-10],例如 RIMOM[11],COMA++[8],Falcon[12].和本文的工作不同的是,这些研究工作只对齐类别,不对齐知识库中的实体.

现有的对于知识库对齐研究主要有两种方法:一是基于概率模型的迭代对齐,二是基于相似性函数的特征匹配. PARIS[13]采用的是基于概率模型的迭代算法进行对齐,PARIS 不仅对齐了实体,还对齐了关系属性以及类别,但是这种方法需要多次迭代计算直至最后收敛,因此需要计算的时间很长. SiGMa[14]采用的基于相似性函数的特征匹配,利用贪心算法进行计算,但是这种方法仅仅对齐了知识库中的实体,对于关系属性和类别都没有进行对齐.

现有的研究例如 CrowdER[15],利用众包进行单纯的实体对齐的研究工作,在这些工作中研究了如何减少预算的问题,但是由于这些实体不存在知识库中,因此,实体的属性相对单一,通常只有一个名字属性,并且实体与实体之间也不存在任何关系,无法运用实体之间的关系以及多属性情况,这些研究工作不能直接运用在知识库的对齐工作中.

3 问题的形式化描述

本文将在这一节引入知识库对齐的定义以及众包的定义.

定义 1. 知识库. 知识库是用于存储现实世界中的实体和和实体之间关系的仓库. 它是由类别 C,实体 E,字面值 L,关系 R,属性 P 组成的集合. 其中,字面值分为字符串、日期和数字三种类型. R 通常表示实体与实体之间的关系($E\times R\times E$),或者是实体与类别之间的关系($E\times R\times C$),P 通常表示实体与字面值之间的关系($E\times P\times L$). 在本文中,我们将用 RDFS(Resource Description Framework Schema)的形式来表示知识库. 一条 RDF 三元组[16]由主语、谓语和宾语组成.

如图 1 所示,给出了知识库当中的一个例子. 在例子中,一共有 3 个实体:“L. Messi”、“FC Barcelona”和“Barcelona”,4 个类别:“Player”、“People”、“Club”和“location”,5 个字面值:“Leo Messi”、“Lionel”、“Messi”、“F. C. Barcelona”和“Barcelona”,4 个属性:“name”、“nickname”、“givenname”和“surname”,4 个关系:“typeOf”、“workIn”、“locateIn”和“subclass”. 知识库中的一个 RDF 三元组〈L. Messi, workIn, FC Barcelona〉表示实体 L. Messi 与实体 FC Barcelona 有关系 workIn. 图 1 右图所示是另一个知识库的一个实例,右图中“Messi”与左图中“L. Messi”指向都是足球运动员梅西,我们需要通过它们周围的属性和与其他实体的关系判断它们是否指向同一实体. 同理于“FC Barcelona”和“Barca”.

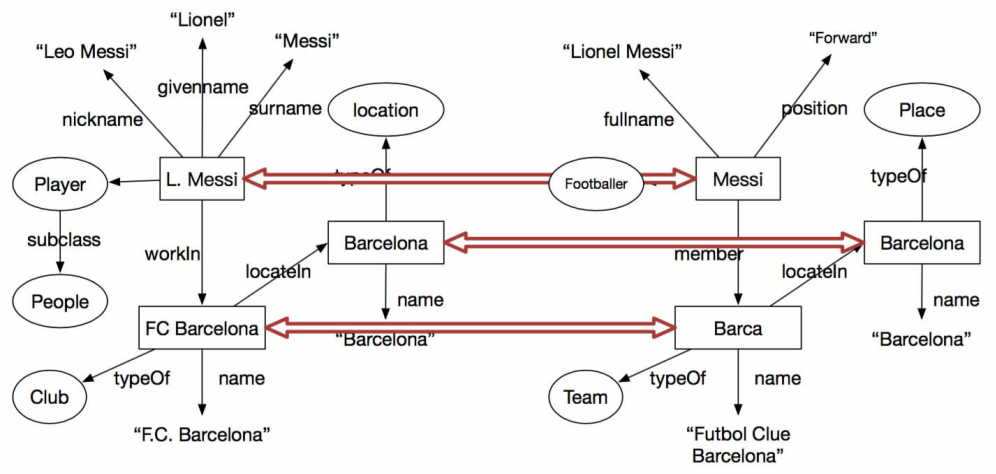


图 1 知识库中的一个实例

下面给出知识库对齐的形式化定义.

定义 2. 知识库对齐. 给出两个知识库 KB1 和 KB2,每个知识库中含有类别、实体、属性、关系和字面值. 知识库对齐是输入阈值,分别对齐两个知识库 KB1 和 KB2 中的类别、实体、属性、关系和字面值.

定义 3. 关系对齐和属性对齐. 给出两个关系 r_1 和 r_2 或属性 p_1 和 p_2，把它们之间的对齐关系归为三类：包含($\supseteq$)，被包含($\subseteq$)和其它($\perp$).

定义 4. 字面值对齐. 字面值分为三种：字符串、日期和数字，分别对齐这三种字面值. 给定一个阈值 δ，根据第 4 节的算法计算出字面值之间的相似度 $similarityScore$，大于等于阈值 δ 的字面值对，则是相似的.

通过属性和关系对齐以及字面值的对齐，我们可以计算出实体之间的对齐关系. 在此之前，我们需要考虑关系和属性与实体之间的指定关系. 我们可以将关系和属性表示成一个指示函数，这个指示函数是为了确定一个实体. 例如“name”这个关系，如果只有一个人叫“Schwarzenegger”，“name”和“Schwarzenegger”就可以确定一个实体，而如果许多人都出生在 1947 年，那么“isBornIn”和“1947-##-##”就不能确定一个实体. 据此，定义局部指示函数：

$$fun(r,y)=\frac{1}{|\{x:r(x,y)\}|} \tag{1}$$

其中，$\langle x,r,y\rangle$ 是一个三元组，x 为主语，y 为宾语. 如果只有一个人叫“Schwarzenegger”，那么 $fun(name,$ “$Schwarzenegger$”$)=1$，而如果 10 个人出生在 1947 年，那么 $fun(isBornIn,$“1947-##-##”$)=0.1$. 由于每一个关系下有许多宾语，每一个关系和其底下任意一个宾语都有一个局部指示函数，那么，关系属性的总指示函数定义为局部指示函数的调和平均数：

$$fun(r)=HM_y(fun(r,y)) \tag{2}$$

指示函数值比较高，就说明一个宾语能指定一个实体，指示函数值比较低，就说明很多实体共享一个宾语. 相似度高的字面值通过指示函数可以确定实体是否是唯一实体，且是否相似. 下面引出实体对齐的定义.

定义 5. 实体对齐. 给出两个实体，通过实体的属性或者与其他实体之间的关系计算出实体之间的相似度.

$$Align(e_1,e_2)=\sum\left(\frac{r_1+r_2}{2},similarityScore(l_1,l_2)\right) \tag{3}$$

其中，$(e_1\times r_1\times l_1)\in KB_1$ 且 $(e_2\times r_2\times l_2)\in KB_2$. 实体与实体之间的相似度是由谓语指示性和宾语的相似性衡量的.

定义 6. 类别对齐. 在类别对齐中，类别与类别之间分为三种关系：包含($\supseteq$)、被包含($\subseteq$)和其它($\perp$)：

$$\Pr(c_1\subseteq c_2)=\frac{I(c_1)\quad I(c_2)}{I(c_1)} \tag{4}$$

$$\Pr(c_1\supseteq c_2)=\Pr(c_2\subseteq c_1) \tag{5}$$

$$\Pr(c_1\perp c_2)=(1-\Pr(c_1\subseteq c_2))(1-\Pr(c_1\supseteq c_2)) \tag{6}$$

$I(c_1)$ 和 $I(c_2)$ 分别代表类别 c_1 和类别 c_2 底下的实体，也即类别 c_1 和类别 c_2 底下相似的实体越多，则包含和被包含的概率越大. 最后本文通过众包确定部分答案推测出全局答案.

众包[17-20]是一个通过网络上大量人力解决问题的做法. 在这个过程中，本文通过部分机器算法减少了过高的人力资源浪费. 在知识库对齐过程中，我们提出一个类别对，需要人决定这对类别对是包含还是被包含或者是没有关系. 本文的目标是通过机器算法选择影响力最大的 k 个问题，放在众包平台①进行回答，获取最佳的答案.

4　知识库对齐框架

本文将在这一节介绍知识库的对齐算法框架，见图 2.

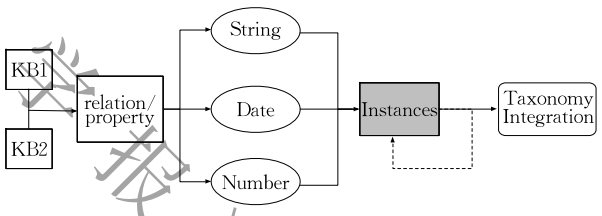


图 2　知识库对齐工作流程

第 1 步，对齐关系属性，并计算指示函数值. 由于知识库的关系和属性不多，我们将手动对齐两个知识库的关系和属性，对于每一对对齐的关系和属性，根据式(1)和(2)计算出指示函数值.

第 2 步，建立索引. 由于属性域不同，分为字符串、日期和数字，对其中一个知识库的每一个属性的所有的三元组根据属性即宾语的不同聚簇，分别建立相应索引，保存在内存之中. 对于宾语是字符串的三元组，建立倒排索引；对于宾语是数字的三元组，建立基于数字的线段树；对于宾语是日期的三元组，建立相应的 DateTrie.

第 3 步，根据索引过滤并获取实体对结果集. 将另一个知识库中的所有三元组根据属性即宾语的不同聚簇，根据第 1 步中对齐的关系属性，将同一个簇

① www.chinacrowds.com

下三元组中实体的所有属性及对应的属性值,通过已经对齐的属性对,在内存中建立的索引中进行搜索,获取不同的候选集,一个实体可能会有若干属性,有字符串表示的,有日期表示的,有数字表示的,通过三种索引获取的结果获选结果集,将相似度叠加,相似度最高的也即获得的候选集交集就是最后的正确实体对.

第 4 步,利用实体结构传递结构性.利用索引快速对齐之后,再利用实体与实体之间的结构性,通过式(3)将对齐的实体对通过关系传递给另一对实体对,也即将对齐的实体看成三元组中的宾语,利用谓语的指示函数,算出主语的相似度,即可获得所有实体对,提高准确性.最后在候选实体对中选取实体对相似度最高的,排除其他不一致候选集,以及排除相似度太低的候选对,获得最后的实体对答案,最后再进行类别对齐.

第 5 步,利用众包对齐类别.给出两个类别 c_1 和 c_2,利用第 4 节算出的所有的实体对以及式(4)~(6)我们能很快速地算出 $\Pr(c_1 \subseteq c_2)$,$\Pr(c_1 \supseteq c_2)$和 $\Pr(c_1 \perp c_2)$通过影响函数,并利用本文提出的算法选出最有影响力的问题,通过剪枝策略减少需要提问的问题数量,再通过已问问题的答案推测出正确的结果.选出的问题放在众宝网上.当问题的数量达到了预算的时候,那么停止算法,剩下的问题直接选取概率大的作为最后答案.

第 5 节主要介绍实体对齐中基于不同的属性值的索引算法,以及实体之间结构性的传递;第 6 节介绍利用众包进行类别对齐.

5 基于字面值的索引算法

本文将在这一节提出基于不同字面值的三种索引.

5.1 基于字符串的倒排索引前缀过滤算法

本小节利用倒排索引进行前缀过滤,加快对齐速度.关于倒排索引,请参考文献[21].

首先给出本文中字符串对齐问题的形式化定义:

假定有字符串 r 和 s,本文使用 $Jaccard$ 系数定义字符串之间的相似度:

$$Jaccard(r,s)=\frac{|r \quad s|}{|r \quad s|} \tag{7}$$

我们认为,对于某一个阈值 δ,当

$$Jaccard(r,s)\geq\delta \tag{8}$$

字符串 r 和 s 是相似的.

在知识库中,字符串 r 和 s 在不同的知识库中存在的三元组分别为:$\langle entity_1, relation_1, r\rangle$ 和 $\langle entity_2, relation_2, s\rangle$,$relation_1$ 和 $relation_2$ 是对齐的,由此,s 在 $relation_1$ 在聚簇下的建立倒排索引中进行搜索过滤,根据算法可以找到字符串 r,也间接找到了 $entity_1$ 与 $entity_2$ 是相似的,字符串 r 和 s 的相似性通过指示函数传递给实体,见式(9).

$$similarity(entity_1,entity_2)=\frac{fun(relation_1)+fun(relation_2)}{2}Jaccard(r,s) \tag{9}$$

在本文所建立的倒排索引中,关键词对应的文档号改为关键词所对应的实体号.

下面本文将举例描述算法过滤获取候选集思想:

首先,我们将字符串 r 和 s 中的单词按照逆文档频率的递减顺序排列,其中逆文档频率 $idf(t)$ 为出现了单词 t 的字符串的个数的倒数.使用字符串前缀过滤获取候选集的过程中,首先需要得到字符串前缀的长度.由 $Jaccard$ 系数的阈值可以得到

$$Overlap(r,s)=|r \quad s|\geq\delta\cdot|r| \tag{10}$$

在字符串 r 和 s 的前缀单词中,至少共享一个单词才不被过滤掉,即

$$Answer(r)=\{s_i|prefix(r) \quad prefix(s_i)\neq\varnothing\} \tag{11}$$

由此可得,前缀长度为

$$\begin{aligned}|prefix(r)|&=\lfloor|r|-(Overlap(r,s)-1\rfloor\\&=\lfloor|r|-(\delta\cdot|r|-1)\rfloor\\&=\lfloor(1-\delta)|r|+1\rfloor\end{aligned} \tag{12}$$

例如,在图 3 中,字符串 r 和 s 根据式(12)可以算出前缀长度为 2,但是前缀单词中没有相同的单词,r 和 s 中的单词由于按照逆文档频率递减排序,所以出现相同的单词个数的最多为后缀的长度 3,根据式(10),$Overlap(r,s)$必须大于等于 4,所以将被过滤掉.而假如 r 和 s 前缀中有一个单词共享,出现相同单词的个数最多为 4,可以认为字符串 r 和 s 是候选相似字符串.接着验证字符串 r 和 s 的 $Jaccard$ 系数是否大于等于阈值 δ.若大于等于阈值 δ,则认为字符串 r 和 s 是相似的,并通过式(8)计算 $entity_1$ 与 $entity_2$ 的相似度;若小于阈值 δ,则剔除该候选项.

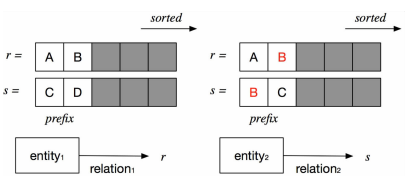


图 3 字符串对齐前缀过滤实例

算法描述：

首先，通过 *initialWordOrder* 函数统计所有单词的逆文档频率，通过 *sortWordOrder* 函数将集合 R 和 S 中所有的字符串按逆文档频率的递减顺序排列，通过 *generatePrefixSets* 函数且根据式(12)可以算出集合 R 中所有字符串的前缀，并通过 *buildInvertedLists* 函数根据前缀建立倒排索引.

接着扫描集合 S 中每个字符串 S_j，可以算出前缀长度 *prefix_length*，通过倒排索引在 *getCandidates* 函数中可以快速得到与 S_j 的前缀重合的字符串 R_i. 在 *getCandidates* 中，首先初始化候选结果集，对前缀长度中每一个 *word*，通过倒排列表快速查找到候选集，加入到结果集中.

最后在 *verify* 函数中通过 *Jaccard* 系数来验证 R_i 和 S_j 的相似性是否大于等于阈值 δ. 是，则加入结果集；不是，则剔除.

给出算法伪代码如下：

算法 1.　基于字符串的倒排索引前缀过滤算法.

Function similarity_join(R,S)
输入：字符串集 R,S
输出：相似字符串对集 P
1. *initialWordOrder*();
2. *sortWordOrder*(R);
3. *sortWordOrder*(S);
4. *generatePrefixSets*(R);
5. *inverted_list*=*buildInvertedLists*(R);
6. FOR EACH S_j in S {
7.　　*prefix_length*=(1−*threshold*) * *len*(S_j)+1
8.　　*candidates*=*getCandidates*(S_j,*prefix_length*);
9.　　*P*.*add*(*verify*(S_j,*candidates*));
10. RETURN P;

Function getCandidates(S_j,*prefix_length*)
输入：字符串 S_j，前缀长度 *prefix_length*
输出：候选结果集 *candidates*
1. *initialCandidates*();
2. FOR (i FROM 0 TO *prefix_length*)
3.　　*word*=S_j[i];
4.　　*result*=*inverted_list*[*word*];
5.　　*candidates*.*add*(*result*);
6. RETURN *candidates*;

5.2　基于日期的 DateTrie 对齐算法

本小节提出一种新的索引结构——DateTrie，该索引结构基于传统的前缀树(即 Trie)索引. 在传统的前缀树中，每个结点存储一个字母，任意一条从根结点到叶子结点的路径都表示一个单词，根结点到非叶子结点都表示一个单词的前缀. 通过前缀树，可以快速找到具有某一个前缀的多有单词.

在知识库中，日期的表示形式为：####-##-##. 例如，1984-##-## 表示 1984 年，具体的月份日期不清楚，1984-07-## 表示 1984 年 7 月，1984-07-24 表示 1984 年 7 月 24 日. 而在日期对齐中，我们使用以下规则定义日期相似度：

$$Similarity(d_1,d_2)=\begin{cases}1, & d_1=d_2\\ 1/12, & d_1\in d_2, d_1\in month, d_2\in year\\ 1/31, & d_1\in d_2, d_1\in day, d_2\in month\\ 0, & d_1\notin d_2\end{cases} \tag{13}$$

例如，1984-07-## 与 1985-##-## 相似度为 0，而 1984-##-## 与 1984-07-## 相似度为 1/12，1984-07-## 与 1984-07-24 相似度为 1/31，1984-07-## 与 1984-07-## 相似度为 1.

和传统前缀树相比，DateTrie 除了能表达某一个日期或者日期前缀以外，还扩充了与日期前缀相关的实体信息. 下面给出 DateTrie 的形式化描述. DateTrie 为一个层次结构，其中每个结点都对应日期前缀和包含的实体列表. 特别地，DateTrie 有具有以下的性质：

(1) DateTrie 只有 4 层节点，第 1 层为根结点，第 2 层为年结点，第 3 层为月结点，第 4 层为日结点，其中，每一个年结点下的月结点最多有 12 个，每一个月结点下的日结点最多有 31 个.

(2) 时间前缀包含性：即父亲结点对应的日期前缀是孩子结点对应的日期前缀.

图 4 给出一个 DateTrie 的实例. 建立的是“isBornIn”的 DateTrie，该 Trie 只有 4 层结点，A 出生在 1984 年，因此挂靠在(‘1984’,A)这个结点上，结点(‘07’,B)这个结点表示 B 出生在 1984 年 7 月，拥有时间前缀包含性，实体只会挂靠在最后一个确切时间粒度上.

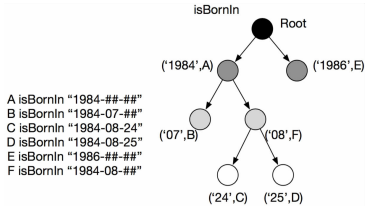


图 4　DateTrie 的一个实例

本文采用自顶向下的算法建立 DateTrie. 对于某一个宾语为日期 d_1 的三元组，我们首先将其日期

分割成“年月日”,指针从 DateTrie 的根开始往下搜索,当已经存在 d_1 年份的结点时,指针跳到该年份的结点上,若不存在,则创造该结点.继续看 d_1 月份,如果月份为“##”,则说明 d_1 只存储了年份,那么该三元组的主语实体挂在现有指针所指向的结点上;如果月份为明确的月份,那么指针跳到该月份的结点上,不存在该月份结点的话,创造该月份结点.日结点同理.

下面给出通过 DateTrie 找到和实体的日期信息相似的实体的算法.

算法描述:给出一个日期,从根结点往下搜索,通过日期当前的数字获取到子结点位置,将子结点里的实体加进候选集,依次从年到月到日,路过的每一个结点中实体及其相似度都加入候选集,当日期的截止在年或者月时,那么当前结点的所有子数下的实体及其相似度全部加入候选集.例如,在图 4 中,“1984-08-##”搜索到的结点为 Root-(‘1984’,A)-(‘08’,F),除了加入(A,1/12 和(F,1)以外,结点(‘08’,F)的子树(‘24’,C)和(‘25’,D)中的(C,1/31)和(D,1/31)也全部加入候选集.

给出算法的伪代码:

算法 2. 基于日期的 DateTrie 的搜索算法.

```
Function startWith(String date)
输入:字符串 date
输出:候选结果集 candidates
1. dates[]=date.split("-");
2. node=root;
3. FOR (EACH num IN dates)
4.    IF (num=="##")
5.       candidates.addAll(node.sons.entities);
6.       BREAK;
7.    pos=num;
8.    IF (node.son[pos]==null) RETURN candidates;
9.    candidates.addAll(node.son[pos].entities);
10.   node=node.son[pos];
11. RETURN candidates;
```

5.3 基于数字的线段树对齐算法

本小节提出一种新的索引结构——基于数字对齐的线段树.线段树是一种二叉搜索树,与区间树相似,它将一个区间划分成一些单元区间,每个单元区间对应线段树中的一个叶结点.对于线段树中的每一个非叶子结点$[a,b]$,它的左儿子表示的区间为$[a,(a+b)/2]$,右儿子表示的区间为$[(a+b)/2+1,b]$.因此线段树是平衡二叉树,最后的子节点数目为 N,即整个线段区间的长度.使用线段树可以快速的查找某一个节点在若干条线段中出现的次数,时间复杂度为 $O(\log N)$.

在知识库中,数字之间的相似度定义如下:

$$diff(num_1,num_2)=\frac{1}{1+\frac{100\cdot|num_1-num_2|}{num_1}} \tag{14}$$

当 $diff$ 值小于给出的阈值 $\delta(\delta\in(0,1))$,则认为两个数字是相似的.

据此式(12)可以计算出 num_1 的上界和下界:

$$ubound(num_1)=\frac{101-\delta}{100}\cdot num_1 \tag{15}$$

$$lbound(num_1)=\frac{99+\delta}{100}\cdot num_1 \tag{16}$$

由上界和下界可以组成一条线段,凡是落在该线段里的点,都认为两个数字是相似的.

和传统的线段树相比,由于线段的上界和下界是小数,并且不知道确切大小,所以需要将所有数字的上界和下界排序,并且与整数做离散化的一一对应.除此以外,由于在本文的线段树中,不是需要查找某一个结点在若干线段出现的次数,而是需要获取与该线段对应的数字相关的实体,因此,每一个线段中还扩充了相应的实体信息.

下面给出基于数字的线段树的形式化定义.线段树是一种平衡二叉搜索树,其中每个结点都表示某一个区间和包含的实体列表.特别地,该线段树有具有以下的性质:

(1)线段树中的每一个非叶子结点表示一个区间$[a,b]$,其中 a 和 b 表示的是在上界和下界排序集合中的排序为第 a 个的数和第 b 个的数,它的左儿子表示的区间为$[a,(a+b)/2]$,右儿子表示的区间为$[(a+b)/2,b]$,叶子结点为$[a,a+1]$,表示的是一个最小区间.

(2)对于一个实体的数字字面值,若结点表示的区间全部被包含在该数字的上界下界中,那么该结点需要扩充该实体信息.

下面给出该线段树一个实例.图 5 为“hasLon-

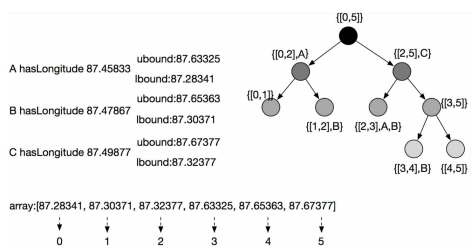


图 5 基于数字的线段树的实例

gitude”的线段树.其中实体 B 的经度的上下界分割成 3 段[1,2],[2,3],[3,4],挂在三个叶子结点上.

本文采用的是自上而下递归的方式建立的线段树.首先计算出某一属性下所有的数字的上下界,并进行排序,用一个数组进行保存.以 0 和数组的大小减 1 作为线段树的根结点的区间递归建立线段树.接着将每一个数字对应的实体信息添加到相应结点上.伪代码如下:

算法 3.　基于数字的线段树对齐算法.

```
Function CreateLineNode
输入:区间值在数组中排序数 leftpos,rightpos
输出:根结点 root
1. LineNode root=NEW LineNode(leftpos,rightpos);
2. IF (rightpos !=leftpos+1)
3.     int mid=(leftpos+rightpos)/2;
4.     root.lchild=CreateLineNode(leftpos,mid);
5.     root.rchild=CreateLineNode(mid,rightpos);
6. RETURN root;
Function Insert
输入:根结点root,数字上下界在数组中的排序数leftpos,
      rightpos,实体 entity
输出:null
1.  IF (leftpos==root.leftpos&&
        rightpos==root.rightpos)
2.     root.entities.add(entity);
3.  ELSE IF (leftpos<=rightpos)
4.     int rmid=(root.leftpos+root.rightpos)/2;
5.     IF (rightpos<=rmid)
6.        Insert(root.lchild,leftpos,rightpos,entity);
7.     ELSE IF (leftpos>=rmid)
8.        Insert(root.rchild,leftpos,rightpos,entity);
9.     ELSE
10.       int mid=(leftpos+rightpos)/2;
11.       Insert(root.lchild,leftpos,mid,entity);
12.       Insert(root.rchild,mid,rightpos,entity);
```

下面给出通过线段树找到和实体的数字信息相似的实体的算法.给出一个数字,自顶向下递归搜索线段树,返回结果集.

伪代码如下:

```
Function getResult
输入:根结点 root,数字 value,上下界集合 arrayList
输出:候选结果集 candidates
1.  initialCandidates();
2.  IF (arrayList.get(root.leftpos)<=value&&
        arrayList.get(root.rightpos)>=value)
3.     candidates.addAll(root.entities);
4.  IF (root.leftpos+1==root.rightpos)
5.     RETURN facts;
6.  int mid=(root.leftpos+root.rightpos)/2;
7.  IF (value<=arrayList.get(mid))
8.     candidates.addAll(getResult(root.lchild,value));
9.  IF (value>=arrayList.get(mid))
10.    candidates.addAll(getResult(root.rchild,value));
11. RETURN facts;
```

6　基于众包的类别对齐算法

本文在这一节讲述利用众包进行类别对齐.本节利用机器的算法选出最有影响力的问题,减少需要提问的问题数量,通过已问问题的答案推测出正确的结果.

6.1　剪枝策略

给出两个类别 c_1 和 c_2,c_1 是 KB_1 中的类别,c_2 是 KB_2 中的类别,利用第 4 节算出的所有的实体对,以及式(4)～(6)我们能很快速地算出 $\Pr(c_1\subseteq c_2)$,$\Pr(c_1\supseteq c_2)$ 和 $\Pr(c_1\perp c_2)$,确定它们之间的关系后,那么可以通过以下策略进行问题的剪枝:

(1)如果 $c_1\supseteq c_2$,那么所有 c_1 的祖先包括 c_1 都是 c_2 的后代包括 c_2 的祖先,可以剪枝的数量为 $\{ans(c_1)\cdot des(c_2)-1\}$.

(2)如果 $c_1\subseteq c_2$,那么所有 c_2 的祖先包括 c_2 都是 c_1 的后代包括 c_1 的祖先,可以剪枝的数量为 $\{ans(c_2)\cdot des(c_1)-1\}$.

(3)如果 $c_1\perp c_2$,那么 c_1 既不是 c_2 的祖先也不是 c_2 的后代,那么 c_1 的祖先也不是 c_2 的祖先或者后代,可以剪枝的数量为 $\{(ans(c_1)+des(c_1))\cdot(ans(c_2)+des(c_2))\}$.

其中 $ans(c)$ 表示 c 的祖先包括自己,$des(c)$ 表示 c 的后代包括自己.

6.2　影响函数

通过计算两个类别 c_1 和 c_2 之间关系的概率,以及剪枝的问题的数量,我们能确定一个问题的影响函数:

$$Inf(q)=\sum \Pr(c_1,c_2)\cdot N_{\Pr(c_1,c_2)} \tag{17}$$

其中,q 指的是类别 c_1 和 c_2 之间的归属关系,$\Pr(c_1,c_2)\in\{\Pr(c_1\subseteq c_2),\Pr(c_1\supseteq c_2),\Pr(c_1\perp c_2))$,$N$ 为类别 c_1 和 c_2 之间的归属关系对应的问题剪枝的数量.

下面给出一个实例.如图 6 所示,可计算出 $\Pr(A\subseteq B)=0.5$,$\Pr(A\supseteq B)=0.7$,$\Pr(A\perp B)=0.15$.标准化之后 $\Pr(A\subseteq B)=0.37$,$\Pr(A\supseteq B)=0.52$,$\Pr(A\perp B)=0.11$.对于 $query(A,B)$ 的不同情

况下的剪枝数量为:3,11,19. 因此,算出 $Inf(q)=0.37\times3+0.52\times11+0.11\times19=9.66$.

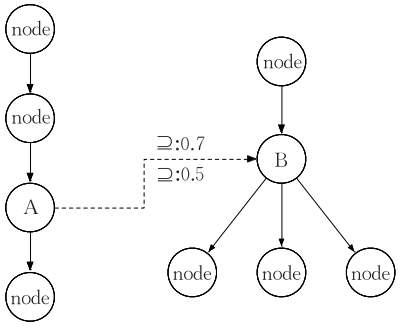


图 6 类别对齐的实例

6.3 动态算法

由于每一个问题的答案都是不确定的,贪心地来获取当前影响函数值最大的问题的答案,利用此答案在候选的问题集里进行剪枝更新问题集里的影响函数值.

下面给出算法的伪代码:

算法 4. 基于众包的动态问题选择算法.

Function query_selection

输入:*query* budget k, two taxonomies T_1, T_2

输出:*query* set Q

1. Initialize *query* set Q based on the entities pair
2. Initialize *queue*
3. FOR EACH *query* IN Q
4. *compute* $Inf(query)$;
5. *queue.enque*(*query*);
6. FOR $i=1$ TO k DO
7. $query\_max=queue.deque()$;
8. Observe *result* of *query_max*;
9. Prune and update *queue*;

7 实 验

7.1 实验设置

数据集:本文选择对比 OAEI 中的 IIMD 基准数据集和知识库 Yago、DBpedia. IIMD 两个知识库都是用 RDFS 来进行表述的,IIMD 是 rdf 格式,Yago 是 ttl 格式,DBpedia 是 nt 格式. 数据集的数据统计见表 1. 由于 Yago 和 DBpedia 中实体都是用的相同的维基百科标识符 URI,所以对于标准答案我们可以通过相同标识符 URI 来快速获取. 获取到的正确的实体对有 2 219 249 对. 而对于关系和属性,由于数量不多,我们采取人工的方式手动对齐,确保了正确性,Yago 和 DBpedia 的关系属性对是一对多的关系. 对于类别对齐,我们通过人工的方式来检查正确性.

表 1 数据统计

数据集	#Facts	#Instance	#Classes	#Relation
Yago	19 805 704	5 208 897	475 661	34
DBpedia	10 800 624	3 071 215	435	121
IIMD	37 846	12 625	218	24

参数设置:本文一共需要三个阈值来控制正确性:字符串文本 *Jaccard* 系数的阈值,数字之间 *diff* 的阈值,以及在类别对齐中需要过滤掉 *similarity* 过小的类别对. 字符串文本的阈值 $\delta=0.8$,数字之间 *diff* 的阈值 $\delta=0.95$,在类别对齐中,过滤掉 *similarity* 小于 0.01 的类别对.

使用语言:Java.

实验环境:处理器为 Intel(R) Xeon(R) CPU E5-26700@2.60 GHz;内存为 128 GB;操作系统为 Linux.

7.2 实验方法

为了说明本文提出的方法的高效性和准确性,实验采取以下的对比方法:

(1) 在 IIMD 上和 Yago-DBpedia 上实验本文提出的方法和使用字符串相同对齐实体(Exact-string)的方法、PARIS 方法,对比 Precision,Recall,*F*-measure 和时间.

(2) 在实体对齐中,有两个阈值需要考虑,一个是字符串文本的阈值,一个是数字对比的阈值,使用不同的阈值,对比 Precision,Recall,*F*-measure 和时间.

(3) 在实体对齐中,本文使用三种索引,在 IIMD 数据集和 Yago-DBpedia 上分别查看候选集缩减率和候选集完整性.

(4) 在类别对齐中,通过设置 budget 的数量的不同,对比剪枝的效果,即减少的类别候选集数量 Reduced Candidate Pairs Number 记忆对比 Precision 的不同. 在判定类别对齐的正确性上,我们在 30 名学生中,判定类别对齐是否正确.

7.3 实验结果

(1) 本文首先在 IIMD 数据集上进行实验,对齐的结果如表 2 所示. 通过实验数据可以看到,使用相同字符串来对齐实体能达到的 *F*-measure 只有 40%,耗时 1 秒,而 PARIS 用了 3 次迭代之后收敛,耗时 6 秒,而本文提出的方法在 2 秒的时间达到 95.7% 的 *F*-measure. 通过结果可以发现本文提出的方法相比于 PARIS 方法耗时更少,并且比使用相同字符

串来对齐实体获得更好的效果.

本文的最终目的是准确而高效地对齐真实的大规模的知识库. 本文在 Yago-DBpedia 上对比了本文方法、PARIS 和 Exact-string 方法. 两个知识库通过对比相同的 URI 得到的真实的相同的实体对达到 2 219 241 对，PARIS 得到的实体对达到 93.5%的准确率和 71.2%的召回率，耗时 839 min，而 Exact-string 方法虽然耗时很短，但是召回率只有 57.2%. 相比于这两种方法，本文的方法达到 90%的准确率和 79.3%的召回率，耗时 25 min，耗时比 PARIS 方法短，而召回率比 Exact-string 高. 不难发现：(1) 本文提出的方法的召回率高一些，主要是由于本文分别用了三种索引来获取所有可能的实体对，并且利用了实体关系来传递相似性，通过这种方式获取尽可能正确的实体对全集，最后再通过剪枝和阈值除去一些错误的实体对，这种方式所得的召回率有明显的改善；(2) 本文提出的方法的准确率稍微低一点，主要是由于在 Yago 中有 520 万的实体，而 DBpedia 中有 307 万的实体，除了正确的 221 万实体，剩余的错误实体对是由于：① 字面值(字符串、日期和数字)的相似导致错误实体对；② 没有考虑字面值的不相似给候选实体对的负面影响. 这可以成为未来的研究方向.

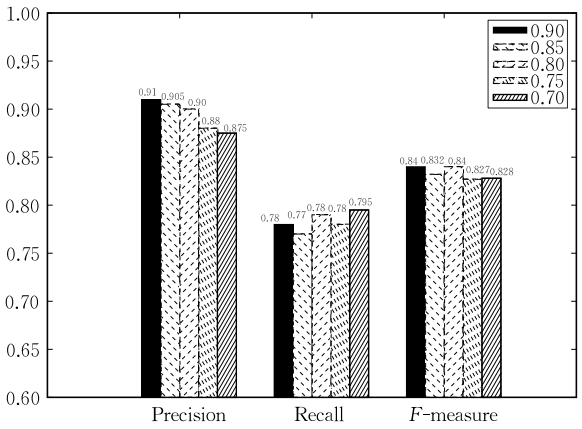
表 2　实体对结果

数据集	方法	准确率/%	召回率/%	*F*-measure/%	时间/min
IIMD	本文方法	98.1	93.5	95.7	0.03
	PARIS	99.7	90.6	94.9	0.06
	Exact-string	100.0	25.2	40.0	0.01
Yago-DBpedia	本文方法	90.0	79.3	84.3	25.00
	PARIS	93.5	71.2	80.8	839.00
	Exact-string	94.3	57.2	79.4	1.00

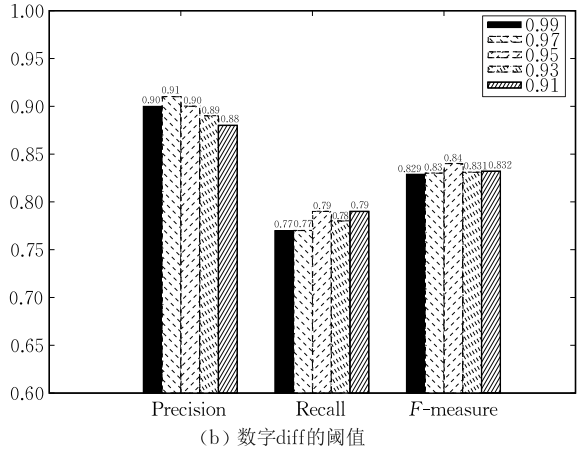
(2) 在本文提出的方法中，实体对齐主要有两个阈值需要考虑，一个是字符串文本的阈值，一个是数字对比的阈值，下面我们想分析在不同的阈值之下本文提出的方法的性能.

字符串文本的阈值. 从图 7(a)中，可以发现太小的阈值，能导致准确率降低，阈值太高，部分正确的实体对也会被过滤掉，导致召回率降低. 从图 8(a)中可以发现，时间随着阈值的升高而降低.

数字 *diff* 的阈值. 和字符串文本阈值相似，从图 7(b)中，可以发现太小的阈值，能产生太对候选实体对，导致准确率降低，而阈值太高，部分正确的

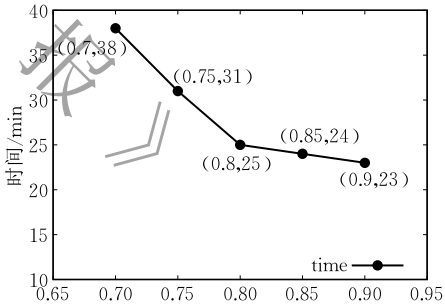


(a) 字符串文本的阈值

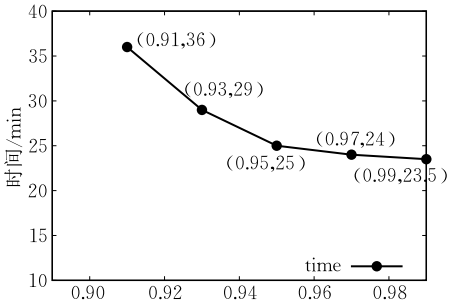


(b) 数字diff的阈值

图 7　基于阈值不同的准确率、召回率和 *F*-measure



(a) 字符串文本的阈值



(b) 数字*diff*的阈值

图 8　基于阈值不同的计算时间

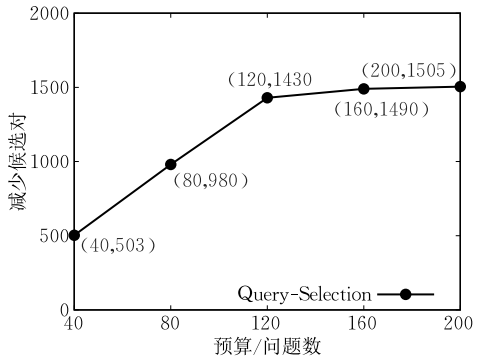
实体对由于数字不够精确也会被过滤掉,导致召回率降低.从图 8(b)中也可以发现,时间随着阈值的升高而降低.

(3)本文使用缩减率(Reduction Radio,RR)[8]来衡量使用索引能减少需要计算匹配实体对数量能力的指标,定义为在不考虑候选对质量的条件下,使用索引减少的实体对与全部待匹配实体对的比值.本文使用候选对完整性(Paris Completeness,PC)[8]来衡量使用索引之后对候选集的质量的指标,它定义为候选集中正确的实体对与所有正确的实体对数量的比值.通过表 3,可以发现,候选集完整性已经非常高,说明质量可以保证,而缩减率在大规模知识库的情况下有些降低,说明候选集的需要降低,因此本文可以适当考虑使用不相似性来减少候选集的大小,这可以成为未来的研究方向.

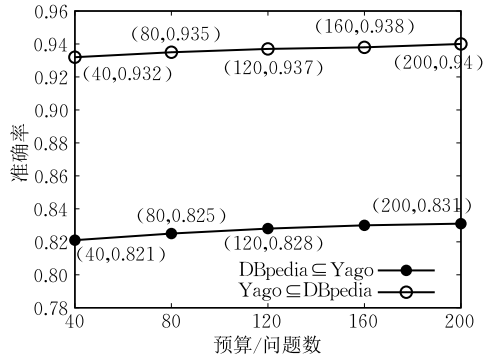
表 3 方法的缩减率和完整性

Dataset	RR	PC	*F*-measure
IIMD	99.9	96.2	98.0
Yago-DBpedia	98.7	87.1	92.5

(4)本文通过预算问题的数量,可以看到减少的候选集中的类别对,以及准确率的变化.通过图 9(a)可以发现,随着预算的增加,减少的类别对越来越多,并且趋于平缓的趋势,说明本文提出的方法还是有效果的,贪心选择影响最大的问题.同样,通过图 9(b)可以发现,随着预算的增加,准确率也在提高,但是也趋于平缓.也同样说明本文提出方法的效果,在一定预算的情况下,减少最多的问题,获取最高的准确性.



(a) 问题减少数



(b) 类别对齐准确率

图 9 基于预算和问题数不同减少候选对和准确率

8 结 论

本文研究了知识库上的对齐算法,设计了三种索引结构对齐字面值,加快了实体的对齐速度,并利用实体与实体之间的结构性提高了准确性.利用众包平台在给定预算的情况下使用机器算法尽可能提高了类别对齐的准确性.实验结果表明,提出的方法在性能和准确性上都有显著的提高.同样,本文有待提高准确率,考虑字面值的不相似给候选实体对的负面影响.这可以成为未来的研究方向.

参 考 文 献

[1] Hoffart J, Suchanek F M, Berberich K, et al. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. Artificial Intelligence, 2013, 194: 28-61

[2] Bollacker K, Evans C, Paritosh P, et al. Freebase: A collaboratively created graph database for structuring human knowledge//Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver, Canada, 2008: 1247-1250

[3] Wu W, Li H, Wang H, et al. Probase: A probabilistic taxonomy for text understanding//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, USA, 2012: 481-492

[4] Auer S, Bizer C, Kobilarov G, et al. DBpedia: A Nucleus for a Web of Open Data. Berlin Heidelberg, Germany: Springer, 2007

[5] Papadakis G, Ioannou E, Palpanas T, et al. A blocking framework for entity resolution in highly heterogeneous information spaces. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(12): 2665-2682

[6] Jean-Mary Y R, Shironoshita E P, Kabuka M R. Ontology matching with semantic verification. Web Semantics: Science, Services and Agents on the World Wide Web, 2009, 7(3): 235-251

[7] Demidova E, Oelze I, Nejdl W. Aligning freebase with the YAGO ontology//Proceedings of the 22nd ACM International Conference on Information & Knowledge Management. San Francisco, USA, 2013: 579-588

[8] Aumueller D, Do H H, Massmann S, et al. Schema and ontology matching with COMA++//Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. Baltimore, USA, 2005: 906-908

[9] Arasu A, Re C, Suciu D. Large-scale deduplication with constraints using dedupalog//Proceedings of the IEEE 25th International Conference on Data Engineering. Shanghai, China, 2009: 952-963

[10] Bhattacharya I, Getoor L. Collective entity resolution in relational data. ACM Transactions on Knowledge Discovery from Data, 2007, 1(1): 5

[11] Li J, Tang J, Li Y, et al. RIMOM: A dynamic multistrategy ontology alignment framework. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(8): 1218-1232

[12] Hu W, Qu Y, Cheng G. Matching large ontologies: A divide-and-conquer approach. Data & Knowledge Engineering, 2008, 67(1): 140-160

[13] Suchanek F M, Abiteboul S, Senellart P. PARIS: Probabilistic alignment of relations, instances, and schema. Proceedings of the VLDB Endowment, 2011, 5(3): 157-168

[14] Lacoste-Julien S, Palla K, Davies A, et al. SiGMa: Simple greedy matching for aligning large knowledge bases// Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Chicago, USA, 2013: 572-580

[15] Wang J, Kraska T, Franklin M J, et al. CrowdER: Crowdsourcing entity resolution. Proceedings of the VLDB Endowment, 2012, 5(11): 1483-1494

[16] Zhuang Yan, Li Guo-Liang, Feng Jiang-Hua. A survey on entity alignment of knowledge base. Journal of Computer Research and Development, 2016, 53(1): 165-192(in Chinese) (庄严，李国良，冯建华. 知识库实体对齐技术综述. 计算机研究与发展，2016，53(1)：165-192)

[17] Udrea O, Getoor L, Miller R J. Leveraging data and structure in ontology integration//Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. Beijing, China, 2007: 449-460

[18] Wang J, Kraska T, Franklin M J, et al. Crowder: Crowdsourcing entity resolution. Proceedings of the VLDB Endowment, 2012, 5(11): 1483-1494

[19] Vesdapunt N, Bellare K, Dalvi N. Crowdsourcing algorithms for entity resolution. Proceedings of the VLDB Endowment, 2014, 7(12): 1071-1082

[20] Fan J, Li G, Ooi B C, et al. iCrowd: An adaptive crowdsourcing framework//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne, Australia, 2015: 1015-1030

[21] Cutting D, Pedersen J. Optimization for dynamic inverted index maintenance//Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 1989: 405-411

SHEN Bing-Wen, born in 1992, M. S. candidate. His research interests focus on database.

FENG Jian-Hua, born in 1969, Ph. D., professor. His research interests include database, structural and semi-structural data query, data cleaning and integration, crowdsourcing data management.

Background

A knowledge base (KB) is a technology used to store complex structured and unstructured information used by a computer system. The initial use of the term was in connection with expert systems which were the first knowledge-based systems. Academia and industry have emerged more and more large scale knowledge base. Knowledge base alignment work has been a hot research problem in recent years. Because these knowledge bases are from different data sources, and use different technologies to integrate data, will inevitably lead to knowledge in different knowledge base repetitive and complementary.

Knowledge base alignment is to align different entities, relationships and the classes of entities. However, due to the enormous size of the knowledge base, and large structural differences in the knowledge base, there are many problems and challenges to be resolved in knowledge alignment work. The existing knowledge base alignment research is mainly based on the similarity function to match or based on the probability model for iterative calculation. The iterative calculation method has high accuracy, but the computation time is long, and the similarity function greedy matching can only match the entities in the knowledge base, and there are

some limitations.

Starting from these challenges and three kinds of property values, this paper proposes three indexes which is based on string prefix inverted index, date DateTrie, number segment tree, and use the indicative function to pass through the alignment to entities alignment. Then we use entities structure to improve the accuracy. Finally, we use machines and artificial combination to control certain labor budget and reduce the candidate set of classes pairs, and use crowdsourcing to align classes which maximize accuracy. Crowdsourcing is a specific sourcing model in which organizations use contributions from Internet users to obtain needed services or ideas. In this paper, we contrast this paper's method, PARIS and Exact-string method on Yago-DBpedia dataset. PARIS obtained entity pairs that achieved 93.5% accuracy and a recall rate of 71.2%, which took 839 minutes while the Exact-string method cost time is 1 minute, but the recall rate is only 57.2%. Compared with these two methods, this paper's method achieves 90% accuracy and the recall rate of 79.3%, takes 25 minutes, is shorter than the PARIS method, and the recall rate is higher than that of Exact-string.