基于人物关系的图片搜索系统①

莫桂烽[1,2], 左 春[1,2,3], 曾 炼[4]

[1](中国科学院软件研究所 软件工程技术研发中心, 北京 100190)
[2](中国科学院大学, 北京 100190)
[3](中科软科技股份有限公司, 北京 100190)
[4](视觉(中国)文化发展股份有限公司, 北京 100015)

摘 要: 针对传统的采用关键词搜索人物图片的方式在使用查询关系人语句进行查询时不能识别语义的不足, 设计并实现了一种基于人物关系的图片搜索系统架构. 基于文档共现和句子共现的关系度算法, 从新闻语料中挖掘了人物之间潜在的关联关系. 创建了人物图片的领域本体库, 本体中包含亲属、朋友、同事等共 174 种具有层级结构的人物关系属性. 提供一个面向查询关系人句子的本体库查询接口, 首先基于依存关系树的合并规则从依存句法树提取查询关系人语句的关键组成部分, 然后基于三元组补全算法转换得到 SPARQL 语句, 接着使用 SPARQL 查询人物图片本体库, 实现语义检索. 最后给出实验结果验证系统的可行性和有效性.

关键词: 关系挖掘; 领域本体; SPARQL; 语义检索

Relationship-Based Image Retrieval System

MO Gui-Feng[1,2], ZUO Chun[1,2,3], ZENG Lian[4]

[1](Software Engineering Center, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)
[3](Sinosoft, Beijing 100190, China)
[4](Visual China Group, Beijing 100015, China)

Abstract: Aiming at the deficiency of identify semantic in traditional keyword tosearch characters the way the picture when you use the query statement to query the relationship between people. The paper designs and achieves an image search architecture based on character relationships. We offer an approach to extract the latent relationships between persons from news corpus. Domain ontology library creates pictures of people and in the ontology contains a total of 174 relatives, friends, colleagues and other kinds of character relationship with the hierarchy properties. An ontology-driven query interface for the sentence which provides a query-oriented relationship of one sentence, first We extract the critical component of the query statement for the relative persons which comsolidation rules based on dependency grammar tree from the dependency syntax tree, then generate the SPARQL sentence with the help of triple supplement algorithm, and use SPARQL sentence to query the image ontology library. The semantic retrieval was realized. Finally, the experiment results were given to verify the feasibility and effectiveness.

Key words: relationship mining; domain ontology; SPARQL; semantic retrieval

名人一直都是人们关注的对象, 网民喜欢搜索名人的图片进行欣赏. 目前的搜索名人图片的方式存在着一个不足之处: 由于当前的搜索是以关键词为特征进行匹配搜索的, 如果使用查询关系人语句进行搜索, 搜索结果会存在和目标人物无关的图片. 此外, 搜索搜索成本过高. 本文希望设计一个基于人物关系的图片搜索系统来弥补上述的不足. 该系统面临有如下两个问题.

① 基金项目:“核高基”重大专项(2010zx01045-001-006)
收稿时间:2015-04-29;收到修改稿时间:2015-06-01

查询关系人语句的语义识别: 以关键词的搜索方人语句的语义. 如查询语句“张三的女儿是谁？”查找的是张七七(张七七是张三的女儿)的图片.

关联人物挖掘: 为了让用户在查询名人图片时能浏览更多的图片, 提供和查询目标相关的名人列表是一个很好的方式. 如何基于大规模语料挖掘名人间的关联关系是系统需要解决的问题.

随着语义网技术的不断发展, 语义网的理论日趋成熟, 已经开发出来并供学术界和工业界广泛使用的通用本体知识库有 WordNet、DBpedia 等[1-3]. 近年来, 为了提高检索系统的语义识别能力, 许多研究机构将本体概念引入检索系统中. 文献[4]设计了一个基于本体的信息提取和检索系统, 其检索方式是以关键词进行检索. 文献[5]对本体的检索机制进行了研究, 设计了基于本体的检索流程. 文献[6]从自然语言的角度出发, 研究了面向自然语言的本体查询接口的设计. 本文基于本体思想来构建系统. 接下来将要介绍系统的设计与关键算法, 最后对系统的研究做出总结.

1　系统概述

视觉中国是中国领先的视觉影像产品和服务提供商, 旗下网站提供了丰富的图片资源. 本系统是视觉中国的预研项目, 目的是提高搜索人物图片的能力. 本文设计并实现了以人物关系为基础的图片搜索的系统原型. 系统暂不考虑重名情况, 即认为一个人名对应着一个人. 系统分成本体库创建子系统和本体库查询子系统. 总体框架如图 1 所示.

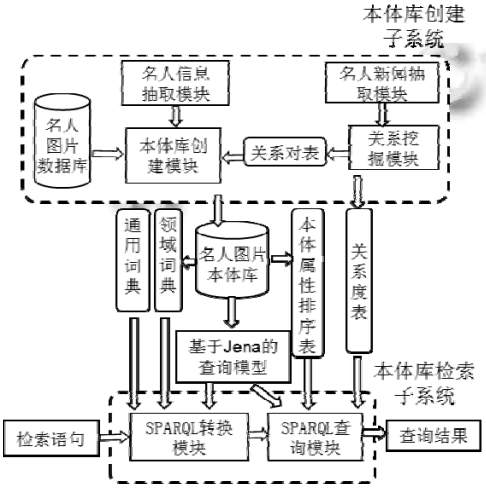

图 1　系统架构图

式不能识别查询语句的语义. 系统需要识别查询关系

本体库创建子系统主要包含名人信息抽取模块、名人新闻抽取模块、关系挖掘模块、本体库创建模块.

名人信息抽取模块从百度百科和互动百科中抽取名人的基本信息(如姓名、年龄等)和关系信息(如父亲、儿子、同事、朋友等).

名人新闻抽取模块从搜狗搜索引擎中获取名人的新闻语料.

关系挖掘模块从新闻语料中挖掘出名人间的关联关系, 得到关系对表和关系度表. 关系对表的每一行是两个存在关联关系的人名. 关系度表的每一行存放着所有和某一人存在关系的人, 按关系度降序排列.

名人图片数据库是已有图片数据库中标注了人名的图片的子集, 图片的信息有图片 ID、图片描述、图片标题、相关图片以及相关人物等.

本体库创建模块对获得的名人基本信息、关系信息、图片信息进行处理, 创建名人图片本体库. 本体库存储了名人和图片的基本信息、名人和名人的关系信息、名人和图片的关联信息以及图片和图片的关联信息, 其中对名人的关联关系进行了详细的划分.

通用词典收录了 386211 个词条, 每一词条都含有词性、词频标注.

领域词典是从本体库中提取得到, 包括本体中定义的类名、实例名、属性名.

本体属性排序表是从本体库中提取各个类的属性并对属性的优先级进行排序得到.

基于 Jena 的本体模型是通过 Jena 提供的方法加载本体库到内存得到, 模型包含了本体中类、属性、实例的名字到 URI 的映射表等资源.

本体库查询子系统是一个 B/S 结构的系统, 用户可通过浏览器输入查询语句进行搜索, 并通过浏览器查看搜索结果. 子系统主要包含 SPARQL 转换模块和 SPARQL 查询模块. 名人图片本体库以基于 Jena 的本体模型的形式为查询子系统库提供数据资源.

SPARQL 转换模块加载通用词典和领域词典, 将查询关系人语句经过分词、依存句法树分析、语义解释处理得到 SPARQL 语句.

SPARQL 查询模块执行 SPARQL 语句对本体库进行查询得到中间结果, 依据属性排序表和关系度表对中间结果进行排序处理, 得到最后的结果.

2 各子系统的关键技术

2.1 本体库创建子系统

2.1.1 本体库创建

本体库是系统的重要组成部分, 由基于描述语言(DL)的 OWL[7]语言进行描述. 2002 年, OWL 正式成为 W3C 推荐的 Web Ontology[8,9]表示语言.

本文依据自顶向下的原则创建本体库. 具体流程如下: ① 概念抽取, 设计了两个类: 人、图片. ②属性定义, 经过统计分析, 本文定义了名字、性别、年龄、职业等共 95 个人物数据属性. 定义了亲人、朋友、同事、合作等共 174 个人物关系属性. 定义了图片 ID、图片尺寸、图片标题等 8 个图片数据属性. 定义了相关图片、相关人物这 2 个的人物和图片间的对象属性. ③创建实例, 本文使用 Jena 提供的 API 来创建实例以及实例间的关系, 存储到 RDF 文件中.

本文建立了丰富的具有层次结构的人物关联关系, 部分关系如图 2 所示. 例如朋友关系分成战友、搭档、伙伴、对象, 对象关系分成男朋友、女朋友的关系. 在搜索人物时, 用户可以通过输入查询人物的关系人语句进行查询, 例如输入“张三的女儿是谁? ”的查询语句进行确定的名人图片查找. 也可根据定义的关系的层次结构进行查询扩展. 如查询张三的亲属时, 能够查询出他的妻子、女儿、弟弟、父亲等.

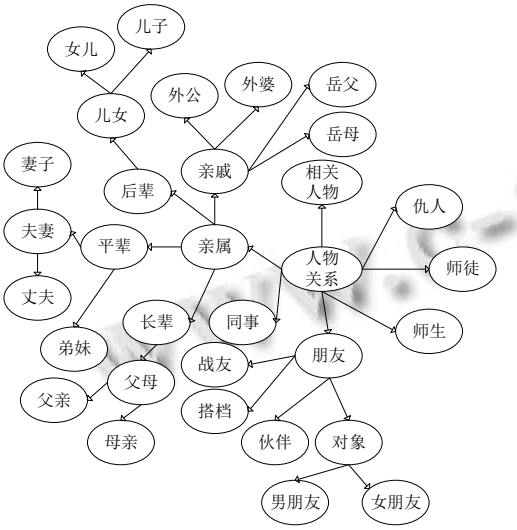

图 2 人物关系层次结构图

2.1.2 关系挖掘

经过统计发现, 53800 个人名中只有 15231 个人名采集到关系信息, 需要进一步挖掘名人间的潜在关系, 并以“相关人物”的属性存储到本体库中. 文献[10]依据人物同时出现的网页或句子的次数来计算人物的关联度, 未考虑人物在句子中出现的位置对人物关系的影响. 本文对文献[10]提出的人物关系算法做出了改进, 给出综合文档共现和句子共现的关系计算方法.

基于文档共现的关联度的计算思想: P_i 和 P_j 两个人共同出现的文档数量和出现 P_i 的数量的比值能体现出 P_j 对于 P_i 的关联度, 该比值和关联度成正比. 计算模型如下:

$$RC(i,j)=\frac{C(P_i,P_j)}{C_i} \tag{1}$$

其中, $C(P_i, P_j)$ 为 P_i 和 P_j 共同出现的文档数; C_i 为 P_i 出现的所有文档数; $RC(i, j)$ 为人物 P_j 对于 P_i 的关联度.

基于句子共现的关联度的计算思想: 一个句子中 P_i 与 P_j 的位置距离 $Distance_{ij}$ 能反应出二者的关联度大小, 和关联度成反比.

假设出现 P_i 句子集合为 S_i, 集合中出现 P_j 的句子集合为 S_{ij}. P_i 在句子 S 中的位置为 Loc_{si}, P_j 在句子 S 中的位置为 Loc_{sj}, P_j 和 P_i 的距离为 $|Loc_{si}-Loc_{sj}|$, S 的长度为 L_s, 计算模型如下:

$$RS(i,j)=\frac{\sum_{s\in S_{ij}}\frac{L_s}{|Loc_{si}-Loc_{sj}|}}{\sum_{s'\in S_i}\sum_{k\in s'}\frac{L_{s'}}{|Loc_{s'k}-Loc_{s'k}|}} \tag{2}$$

其中 $L_s \big/ |Loc_{si}-Loc_{sj}|$ 式子和 P_i 与 P_j 在句子 S 中的关系度成正比, $RS(i, j)$ 表示在 P_j 对于 P_i 的关系度的大小.

因为基于文档共现的关系比基于句子的关系可信度偏弱, 综合考虑式(1)和式(2)的度量方式, 我们给出了最终的关系模型:

$$R(i,j)=\alpha RC(i,j)+\beta RS(i,j) \tag{3}$$

其中 $\alpha+\beta=1$, $0\le\alpha\le1$, $0\le\beta\le1$, 取 $\alpha=0.25, \beta=0.75$.

2.2 本体查询子系统

查询子系统的流程如图 3 所示, 其维护了一个面向查询关系人语句的本体库查询模型, 主要分成 SPARQL[11,12]转换模块和 SPARQL 查询模块.

子系统识别的查询关系人语句类型可分为三类:

正向查询, 如“张三的女儿是谁? ”.

反向查询, 如“张七七是谁的女儿？”.

多重关系查询, 如“张三的妻子的父亲的朋友的同事是谁？”.

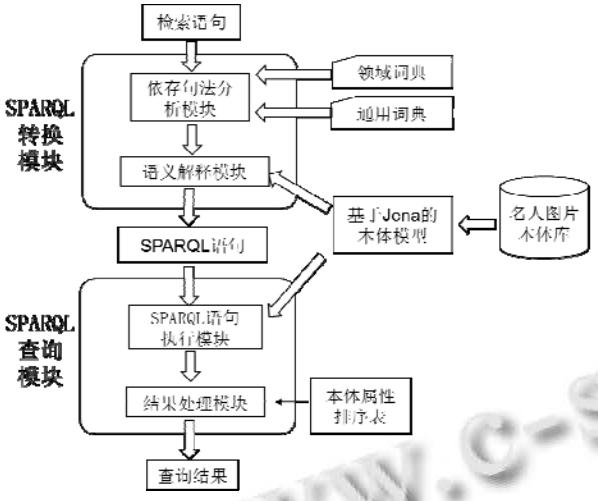

图 3　查询子系统流程图

2.2.1 SPARQL 转换模块

将查询语句转换成 SPARQL 语句的常用方法是基于模板匹配[13], 这类方法预先对大量的问句实例进行分析, 建立结构化的问句模板库, 并对每一类模板人工编写 SPARQL 查询语句. 查询时, 计算查询语句和问句模板的相似度, 将查询语句对应到某一个查询模板, 然后构造出 SPARQL 语句. 这类的方法在引入新的查询类型的语句时, 需要手工添加新的问句模板, 编写新的 SPARQL 查询语句. 另一种方法是通过对查询语句进行句法分析和语义解释[14]转换成 SPARQL 语句, 不需要预先建立问句模板库就能支持不同类型的问句的转换. 本模块采用了语义分析和语义解释的方式获取 SPARQL 查询语句.

SPARQL 转换模块由依存句法树分析和语义解释这两大子模块组成. 一条查询语句经过 SPARQL 转换模块处理后, 得到一条或者多条 SPARQL 查询语句.

2.2.1.1 依存句法树分析模块

定义 1. 自然语言三元组(NLPTriple): 存在依存关系的两个词根据依存关系树的合并规则合并得到一个三元组, 其形式是:

<Subject, Predicate, Object>

其包含了主语(Subject)、谓语(Predicate)和宾语(Object)三个部分, 它们都是关键词. 一个查询三元组的例子: <张三, 的, 女儿>.

定义 2. 自然语言查询对象(NLPQuery): 查询语句的依存关系树经过基于依存关系树的合并算法处理后, 得到一个 *NLPQuery* 对象. 其中, *NLPQuery* 包含一个 *NLPTriple* 列表和一个目标词 *targetWord*, 其结构为:

{

target: *targetWord*,

triples: *NLPTriple0, NLPTriple1* …

}

模块中, 一条查询语句依次经过分词、同义词转换、构建依存句法树、获取依存关系和基于依存关系树合并算法处理得到自然语言查询对象.

系统使用分词器进行分词, 分词时将领域词典和通用词典加入分词器可以提高分词的准确率. 分词结果经过同义词转换后作为输入, 调用 Stanford Parser 的 LexicalizedParser.loadModel 方法加载模型, 执行 apply 方法获得查询语句的句法树.

以句法树作为输入, 创建 Stanford Parser 的 ChineseGrammaticalStructure 对象, 调用其中的 typedDependenciesCCprocessed 方法得到依存关系.

以“张三的妻子的表伯父是谁？”为例, 得到的依存关系如下:

assmod(妻子-3, 张三-1)

case(张三-1, 的-2)

assmod(表伯父-5, 妻子-3)

case(妻子-3, 的-4)

nsubj(谁-7, 表伯父-5)

cop(谁-7, 是-6)

root(ROOT-0, 谁-7)

依存关系中, 前者是支配词, 后者是依存词. 依存关系词就是指依存词对支配词的依赖类型. Stanford Parser[15]给出了至少 45 种中文语句的依存关系, 常见的依存关系标记如表 1.

表 1　常见依存关系

依存关系标记	解释
root	根节点
nsubj	名词主语
punct	标点符号
nn	同位语
conj	连接词
amod	副词修饰语
dobj	直接宾语
pre	介词

• 基于依存关系树的合并算法

算法思想: 一个查询语句可以得到一棵依存关系树, 除了根节点不依赖其它节点外, 依存关系树中其它的节点都有且仅有一个父节点. 关系树的节点可以依据子节点和父节点的依存关系类型进行合并, 生成 0 个或一个 *NLPTriple*, 删除子节点, *NLPTriple* 归属到父节点. 通过节点的不断合并, 最终可从依存关系树中得到一个三元组列表.

父节点和子节点的基于依存关系类型的合并规则如表 2.

表 2 部分基于依存关系树的合并规则

father	child	type	result
dep	nn	TRIPLE	<child , ,father>
top	rcmod	TRIPLE	<father , ,child>
dobj	Rcmod	NOOBJ	<father ,child , >
nsubj	nn	TRIPLE	<child , ,father>
prep	pobj	TRIPLE	<child , ,father>
rcmod	dobj	NOSUBJ	< ,father ,child>
rcmod	cpm	NORESULT	null
assmod	case	NORESULT	null
assmod	assmod	TRIPLE	<child , ,father>
nsubj	assmod	TRIPLE	<child , ,father>
root	nsubj	TRIPLE	<child , ,father>

表中的 father 列是父节点的依存类型, child 列的是子节点的依存类型, type 列的是子节点和父节点合并后得到的三元组类型, 可分为四种类型: NORESULT(节点合并后无结果)、TRIPLE(合并后生成三元组)、NOSUBJ(生成的三元组缺少主语)、NOOBJ(生成的三元组缺少宾语).

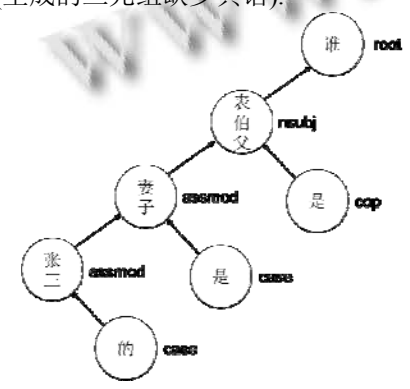

图 4 示例的依存关系的树形结构

如图 4 所示, 父节点“张三”的依赖类型为 assmod, 子节点“的”的依赖类型为 case, 合并后的三元组为空, 节点“的”被舍弃. 父节点“妻子”的依赖类型为 assmod, 子节点“张三”的依赖类型为 assmod, 合并后结果为<张三,妻子>.

基于依存关系树的合并算法流程如下:

① 读取依存关系树 *PTree*.

② 找出 *PTree* 的根节点 N_0, N_0 为依存关系标识为 *root* 的节点.

③ 获取 $N_i(i = 0, 1...)$节点的子节点列表, 从左往右依次对子节点 Sub_i 进行如下处理:

1) 如果 Sub_i 还有子节点, 递归执行③.

2) 如果 Sub_i 没有子节点, 根据依存关系的合并规则, 对 Sub_i 和 Sub_i 的父节点的依存关系进行合并操作, 生成自然语言三元组 $Temp_i$, 从 *PTree* 中删除节点 Sub_i, $Temp_i$ 加入到 Sub_i 的父节点的三元组列表中.

④ 从生成的三元组列表中确定查询语句的目标词. 若只有一个节点, 则根节点是目标词. 否则需要对合并后的三元组列表做出处理:

1) 根节点是代词, 先找到代词在三元组列表中的 *NLPTriple*. 如果代词在 *NLPTriple* 中是在宾语位置, 则取 *triple* 的主语作为目标词.

2) 根节点是名词, 通常这类树是一颗左子树, 如果三元组列表存在代词, 取代词作为目标词. 如果三元组中不存在代词, 取根节点所在三元组的宾语作为目标词.

3) 根节点是其他的词性, 当存在一个缺少主语的三元组< , P , O >和一个缺少宾语的三元组< S_1 , P_1 , >, 将两个三元组删除, 取< S_1 , P_1 ,O >作为新的三元组加入列表, O 作为目标词. 当只存在一个缺少宾语的三元组的时候< S , P , >, 取 S 作为目标词, 将该三元组删除.

⑤ 代词消解, 如果生成的三元组的目标词的词性是代词.

1) 代词是三元组 *triple* 的宾语时, 将代词映射到本体库的词 *word*(如“谁”→“人”). 如果 *word* 的词性是对象属性, 将目标词设为 *word*, 三元组中的 *target* 设为 *word*. 否则将目标词设为三元组的主语, 将 *triple* 删除.

2) 代词是三元组 *triple* 的主语时, 将代词映射到本体库的词 *word*. 如果 *word* 在本体中的类型是类, 当

triple 的宾语是本体的实例, 将目标词设为 *triple* 的宾语, 删除 *triple*. 当 *triple* 的宾语不是本体的实例, 则将目标词设为 *word*, *triple* 的主语设为 *word*.

⑥ 三元组列表重排序. 目标词在三元组 *triple* 中. 当三元组的个数大于一个, 如果列表中邻接的两个三元组中前一个三元组的宾语不是后一个三元组的主语, 则将三元组列表重新排序. 若目标词是主语, 链表以 *triple* 为首个三元组, 不断往后邻接. 若目标词是宾语, 则链接以 *triple* 为列表末尾三元组, 不断往前邻接.

示例经过基于依存关系树的合并算法处理后, 可以得到一个自然语言查询对象:

{
target: 表伯父,
triples: <张三, , 妻子>,
<妻子, , 表伯父>
}

2.2.1.2 语义解释

定义 3. 本体元数据(OntoTripleMeta): 语义解释中, *NLPTriple* 中的每一个词都转换成一个本体元数据, 其形式为:

<*Word*, *Uri*, *Type*, *TripleType*>

其中 *Word* 是词; *Uri* 是词在本体中的唯一标志(当词是本体中的对象、属性、实例的时候, *Uri* 不为空, 否则 *Uri* 为空); *Type* 则是标识 *Word* 在本体中的类型(类 Class、数据属性 DatatypeProperty、对象属性 ObjectProperty、实例 Individual、文字 Literal); *TripleType* 是 *Word* 在本体三元组中的类型(实体 ENTITY、变量 VARIABLE、公共元素 MEDIUM、文字 LITERAL).

定义 4. 本体三元组(OntoTriple): 自然语言三元组经过语义解释会转换成本体的对应的数据结构, 其形式为:

<*Subject*, *Predicate*, *Object*>

其中每一个元素都是一个 OntoTripleMeta.

定义 5. 本体查询对象(OntoQuery): 自然语言查询对象 NLPQuery 经过语义解释转换后得到对应的一个本体查询对象, 其形式为:

{
target: *OntoTripleMeta*,
triples: *OntoTriple*, *OntoTriple2* …
}

查询语句通过依存句法树分析模块处理后得到自然语言查询对象 *NLPQuery*. *NLPQuery* 将通过三元组补全、实体空间映射以及 SPARQL 生成这三个步骤得到 SPARQL 查询语句.

• 三元组补全算法

算法思想: 本体库中最基本数据存储格式是三元组<*Subject*, *Predicate*, *Object*>. SPARQL 的查询是基于图模式匹配的, 以三元组<*S*, *P*, *O*>的方式进行查询, SPARQL 查询原理是通过已知 *S*、*O*、*P* 中的一个或两个的元素来确定其他的元素的值. *S* 是本体的 Individual; *P* 指本体中的属性(DatatypeProperty 或 ObjectProperty); *O* 是 *S* 对应的属性值. 当 *P* 是 DatatypeProperty 时, *O* 是一个 Literal; 当 *P* 是 ObjectProperty 时, *O* 则是一个 Individual.

接收到的 *NLPQuery* 的数据元素都是纯文本词汇, 系统根据 *NLPTriple* 的 *Subject* 和 *Object* 元素在本体的类型, 对三元组的组成进行补全, 最终将 *NLPTriple* 转换成 *OntoTriple*, *NLPQuery* 转换成 *OntoQuery*.

NLPQuery 传递过来的一个三元组为<*S*, , *O*>, 根据 *S* 和 *O* 对应到本体的元素类型, 可以确定缺失的是哪个元素, 在转换成的 *OntoTriple* 中补全该元素, 并标明 *OntoTriple* 的类型.

首先确定 *S*、*O* 两个词映射到本体的数据的数据类型, 映射的优先顺序是 Class > ObjectProperty > DatatypeProperty> Literal. 设 *S*、*O* 在补全后转换成 *S′*、*O′*, 补全的元素为 *N′*, *N′*的 TripleType 为 VARIABLE. 根据 *S*、*O* 的类型的组合进行三元组补全:

① *S*、*O* 都是 Individual, 缺失的元素是谓语, 得到的三元组为< *S′*, *N′*, *O′* >;

② *S* 是 Individual, *O* 是 ObjectProperty, 缺失的是宾语, *O* 在 *OntoTriple* 中作为谓语存在, 得到的 OntoTriple 为< *S′*, *O′*, *N′* >;

③ *S* 是 Individual, *O* 是 DatatypeProperty, 缺失的是宾语, *O* 在 *OntoTriple* 中作为谓语存在, 得到的 *OntoTriple* 为< *S′*, *O′*, *N′* >;

④ *S* 是 ObjectProperty, *O* 是 Individual, 缺失的是主语, *S* 在 *OntoTriple* 中作为谓语存在, 得到三元组为< *N′*, *S′*, *O′* >;

⑤ *S* 是 ObjectProperty, *O* 是 ObjectProperty, 缺失

的是宾语, *O'*在 OntoTriple 中作为谓语存在, 得到的 *OntoTriple* 为< *S'*, *O'*, *N'* >, *S'*的 TripleType 是变量;

⑥ *S* 是 ObjectProperty, *O* 是 DatatypeProperty, 不妨定义其缺失了宾语, *O'*在 *OntoTriple* 中作为谓语存在, 得到的 *OntoTriple* 为< *S'*, *O'*, *N'* >, *S'*的 TripleType 是变量;

⑦ *S* 是 ObjectProperty, *O* 是 Class, 缺失了宾语, *O'*作为谓语存在, 得到的 *OntoTriple* 为< *S'*, *O'*, *N'* >, *S'* 的 TripleType 是变量;

⑧ *S* 是 DatatypeProperty, *O* 是 Literal, 缺失了主语, *S'*作为谓语存在, 得到的 *OntoTriple* 为< *N'*, *S'*, *O'* >, *S'*的 TripleType 是变量;

⑨ *S* 是 Class, *O* 是 Individual, 缺失了谓语, 得到的 *OntoTriple* 为< *S'*, *N'*, *O'* >, *S'*的 TripleType 是变量;

⑩ *S* 是 Class, *O* 是 ObjectProperty, 缺失了宾语, 得到的 *OntoTriple* 为< *S'*, *O'*, *N'* >, *S'*的 TripleType 是变量;

⑪ 其他情况不能补全.

通过三元组补全算法, *NLPQuery* 转换成一个 *OntoQuery* 对象, 接下来通过实体空间映射和 SPARQL 生成两大步骤生成 SPARQL 查询语句.

OntoQuery 中的元素在本体中对可能对应着多个实例, 通过实体空间映射找出 *OntoQuery* 的每一个元素映射到本体的所有实例, 得到一个实体空间映射表.

根据词汇空间映射表构造 SPARQL 查询语句, 依据 OntoQuery 的类型, 相应构造 Construct 或 Select 的查询语句, 一个 *OntoQuery* 可以生成多条 SPARQL 语句.

2.2.2 SPARQL 查询模块

SPARQL 查询模块分为 SPARQL 语句执行模块、结果处理模块.

SPARQL 语句执行模块根据 SPARQL 语句的类型调用 Jena[16]开发包的 ARQ API 中的 QueryExecution 的 execSelect 或者 execConstruct 方法对本体库进行查询, 获取结果.

结果处理模块对查询结果按照属性优先级和人物关系亲密度进行排序处理, 并对查询结果进行封装.

3 系统实验结果与分析

3.1 关联关系挖掘实验

通过对 127543 篇网页文档挖掘, 得到 36629 个名人之间的关系, 总共 627090 对关联关系. 从挖掘结果中随机的选取了 10 个人的前 10 个关联关系人与百度搜索推荐的关联人物进行比较, 选取的人分别是陆某曼、李某白、刘某、陈某、韩某英、成某、左某棠、胡某斌、刘某华、李某(试验结果中对涉及的名字模糊处理, 使用“某某”的方式来代替真实的名字). 经过对比分析, 关系挖掘算法得到的陆某曼、李某白、刘某、陈某、韩某英、成某、左某棠的关联人物结果较好, 而胡某斌、刘某华、李某推荐结果不如百度给出的结果, 但是也具有一定的合理性. 我们可以认为本文提出的综合文档共现和句子共现的人物关系挖掘算法具有一定的准确性.

3.2 SPARQL 转换模块实验

经过试验, 依存句法树分析模块能够识别表 3 中的查询关系人语句, 语义解释模块能将它们转换为 SPARQL 查询语句, 达到了 SPARQL 转换模块识别并转换查询关系人语句成 SPARQL 语句的目标.

表 3 依存句法分析示例

示例一	输入: 张三
	输出: {target:张三, triples: }
示例二	输入: 张三是谁?
	输出: {target:张三, triples: }
示例三	输入: 张三的女儿是谁?
	输出: {target:女儿, triples:<张三, , 女儿>}
示例四	输入: 谁的民族是汉族?
	输出: {target:人, triples:<人, , 民族>, <民族, , 汉族>}
示例五	输入: 谁的女儿是赵七?
	输出: {target:女儿, triples:<人, , 女儿>, <女儿, , 赵七>}
示例六	输入: 张三是谁的母亲的朋友?
	输出: {target:人, triples:<人, , 母亲>, <母亲, , 朋友>, <朋友, , 张三>}
示例七	输入: 谁的朋友的妻子的小叔子是李四?
	输出: {target:人, triples:<人, , 朋友>, <朋友, , 妻子>, <妻子, , 小叔子>, <小叔子, , 李四>}

3.3 系统查询实例

在系统中输入“刘某欢的女儿是谁”, 能准确得到刘某丝的图片列表及其基本信息(刘某丝是刘某欢的女儿), 系统可正确执行正向查询关系人语句. 同时给出刘某丝的相关人物, 方便用户浏览更多的名人图片.

输入“刘某欢是谁的丈夫”, 能查询到卢某璐的信息(卢某璐是刘某欢的妻子), 系统可正确执行反向查询关系人语句.

输入“刘某欢的亲属”, 能查询到刘某丝、卢某璐等人的图片等信息, 系统可执行扩展关系查询.

输入“刘某欢的妻子的父亲是谁”, 能查询到卢某芳的信息(卢某芳是卢某的父亲), 系统可执行多重关系查询.

3.4 查询本体库的性能测试

表 4 不同规模下本体库查询响应时间(ms)

实体数	人名	简单关系语句	多重关系语句
2000	15	19	25
6000	29	35	39
10000	47	55	61
20000	87	89	104
40000	171	176	189
57000	277	301	300

下图是不同规模下本体库查询响应时间的折线示意图:

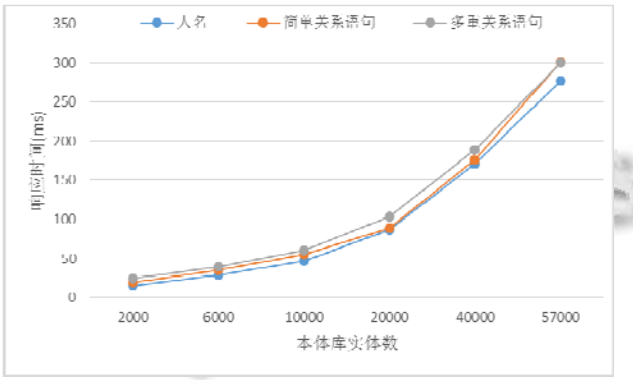

图 5 不同规模下本体库查询响应时间

系统创建的本体库包含了 57630 个人物实体, 66791 对确定类型的人物关联关系, 627090 对一般类型的人物关联关系. 选取了三种典型的查询语句对不同规模的本体库查询进行响应时间, 表 4 和图 5 是实验结果.

随着数据规模的增加, 系统的查询时间也随之增加, 多重关系查询语句耗费时间最多, 人名查询语句的时间最短. 在本体库的实体数达到 57000 个的时候, 查询耗费的时间只有 300 毫秒左右, 这说明系统对本体库的查询具有很好的性能.

4 结语

本文提出的系统模型只是基于人物关系的图片搜索系统的初步结果. 为了进一步提高这种模型的可用性, 还有很多难点需要解决, 比如解决精确识别人物间的关系类型、复杂语句的语义分析、带有条件限制的查询语句转换 SPARQL 语句等问题.

参考文献

1 尚新丽.国外本体构建方法比较分析.图书情报工作,2012, 56(4):116–119.

2 Zhao H, Zhang S, Zhao J. Research of Using Protégé to Build Ontology. ACIS-ICIS. 2012: 697–700.

3 Ra M, Yoo D, No S, et al. The mixed ontology building methodology using database information. Proc. of the International MultiConference of Engineers and Computer Scientists. 2012, 1.

4 Kara S, Alanö, Sabuncu O, et al. An ontology-based retrieval system using semantic indexing. Information Systems, 2012, 37(4): 294–305.

5 许晓林,牛强,林伟.提升机故障知识本体检索机制研究.微电子学与计算机,2012,29(3):64–68.

6 康树鹏.面向语义网的自然语言查询接口研究[硕士学位论文].哈尔滨:哈尔滨工业大学,2014.

7 Matentzoglu N, Bail S, Parsia B. A corpus of OWL DL ontologies. Description Logics, 2013, 1014: 829–841.

8 De Almeida Falbo R, Barcellos MP, Nardi JC, et al. Organizing ontology design patterns as ontology pattern languages. The Semantic Web: Semantics and Big Data. Springer Berlin Heidelberg, 2013: 61–75.

9 Debattista J, Lange C, Auer S. daQ, An ontology for dataset quality information. Linked Data on the Web (LDOW), 2014.

10 邸楠,姚从磊,李晓明.基于中文 Web 社会网络的提取,测量与分析.广西师范大学学报(自然科学版),2007,25(2): 169–172.

11 Ferré S. SQUALL: The expressiveness of SPARQL 1.1 made available as a controlled natural language. Data &

Knowledge Engineering, 2014, 94: 163–188.

12 Kollia I, Glimm B, Horrocks I. SPARQL query answering over OWL ontologies, The Semantic Web: Research and Applications. Springer Berlin Heidelberg, 2011: 382–396.

13 Pradel C, Haemmerlé O, Hernandez N. Natural language query interpretation into SPARQL using patterns. COLD@ ISWC2013. Sydney, Australia. 2013.

14 Habernal I, Konopík M. SWSNL: semantic web search using natural language. Expert Systems with Applications, 2013, 40(9): 3649–3664.

15 De Marneffe MC, Dozat T, Silveira N, et al. Universal Stanford dependencies: a cross-linguistic typology. 9th International Conference on Language Resources and Evaluation. 2014. 4585–4592.

16 Jena. https://jena.apache.org/.