基于知识图谱的批量 Q/A 系统

1. 背景

将知识图谱应用于 *question answer* 领域是非常重要的热门领域之一。但在这个领域中存在两个挑战。

其一是如何快速在 RDF 数据集上高效的执行SPARQL 查询 Q。

其二是如何将一个自然问句 N 准确的转化为 Q。对于一个没有经过专门训练的用户来说，准确的写出 SPARQL 查询是非常困难的一件事。

针对第一个挑战, 目前已经存在大量的解决方法, 包括单机版 RDF engine（RDF3X, gStore）以及分布式查询引擎（gStoreD, triAD, Adpart）

针对第二个挑战也已经有不少解决方法，例如去年发表在 TKDE 上的文章 gAnswer, 在这篇论文中提出了两种将 N 转化为 Q 的方法。

但是事实上，真正的去做基于知识图谱的 Q/A 系统还有一个***第三个的挑战***要面对。“如何高效的处理批量 N 并返回结果”。

通常来说，一个客户端很可能在短时间内接到大量的问句 N。并且这些 N 之间很可能存在公共部分 Q^c，而重复执行这些 Q^c 会导致查询效率的降低。

不过目前已经有一些非常好的工作给出了如何高效的批量处理 SPARQL。例如“Multi-Query Optimization for Subgraph Isomorphism Search“与“Scalable Multi-Query Optimization for SPARQL”。

但是在 QA 系统中寻找公共结构或许还不同于这些工作。

1. 整体架构图以及思路

N： 代表 *question*

S^N： 代表 *question* 的集合

Q： 代表 *Sparql*

R： 代表 *result*

本质上从 N 到 R 的过程是两个函数映射，即(1)和(2)。但是为了更好的贴合实际就有了公式(3)

$$f(N) \rightarrow Q \qquad (1)$$

$$g(Q) \rightarrow R \qquad (2)$$

$$Bg(Q) \rightarrow R \qquad (3)$$

公式 1 指的是将 N 转化为 Q 的方法，**公式 2** 指的是各种查询引擎在 RDF 数据上快速匹配 SPARQL 的方法，**公式 3** 指的是如何处理批量 Sparql 查询。

（1）目前所有的 Q/A 系统要处理 S^N 中所有的 N，都只能逐个执行公式 1 和公式 2 并返回结果。目前还没有考虑批量 Q/A 的工作。

（2）由于已经存在公式 3 的方法，因此我们选择将(1)中的步骤进行简化。先对 S^N 中所有的 N 执行公式 1，再对产生的所有 SPARQL 执行公式 3。这样比较好的利用了公式 2 中的公共结构。但是，公式 1 也可能是存在公共结构。

（3）针对(2)中提出的问题，可能很容易想到添加一个能批量将N转化为Q的公式(4)：

$$\mathrm{Bf(N)} \rightarrow Q \qquad (4)$$

有了公式4就可以先对S^N集合执行公式4，再对生成的Q集合执行公式3。这样能比较好的利用公式1和2中的公共结构，并提升效率。

但是，在这个方法中，其实隐藏了一个类似于多线程的同步动作。即“要等待所有的N都转化为Q之后才能执行公式3”。

那么是否有办法去掉这个等待动作呢？这很重要。因为多线程的同步动作非常的耗费时间。为了去掉这个同步时间，我们给出了下面这种方法。

（4）公式1的执行过程中，会依赖“句法分析树”（denoted as T，这个树能很好的分析N中单词之间的依赖关系以及词语的词性，因此不可或缺）。我们想，最好的应该是合理的找出“句法分析树”中的公共结构T^c。之后分别将（T^c）与(T - T^c)转化为Q^c和Q^-。

假设Q^c首先被生成，Q^c就先执行公式2，获取结果R^c，之后Q^-生成，再对Q^-执行公式2，生成R^-。最后执行$R^c \bowtie R^-$就可以得到最后的结果。

通过这种方法，我们可以避免(3)中同步造成的时间浪费问题。

2. 公式 1 中存在的方法

主要介绍的是 2018 年发表于 TKDE 的文章 gAnswer 中给出的方法。在这篇论文中一共提出了两个框架。其一为关系优先框架，其二为结点优先框架。

句法依存树(denoted as d^s)的定义：

给定一个集合 $R=\{r_1,....,r_R\}$，其中每个元素表示一种依存关系（譬如 SBV，ATT，VOB 等），一个句子的依存树是一棵有向树 $G=(V,A)$，满足以下条件：

（1） $V=\{0,1,...,n\}$，V 是依存树中**顶点**的集合；

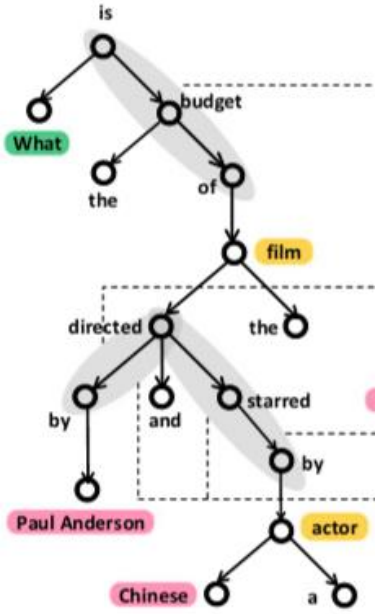
（2） $A\subseteq V\times R\times V$，$A$ 是依存树中**依存弧**的集合。

V 是顶点集合，用非负整数表示，V 中每个顶点依次与句子 s 中的单词 w_i 相对应（ROOT 标号为 0）。A 为依存弧集合，用三元组 (w_i,r,w_j) 表示，w_i **和** w_j **是顶点，** r **表示** w_i、w_j **顶点间的依存关系。**本文规定在三元组 (w_i,r,w_j) 中，依存弧由 w_i 指向 w_j，即 w_i 是 w_j 的头结点（父亲结点），r 是 w_j 的依存关系类型。

例子：

What is the budget of the film directed by Paul Anderson and starred by a chineses actor?

其简化版句法依存树如下，这个句法分析树只保留了结构，没有显示每个词的词性，以及词语之间的关系。



接下来简短的描述两种算法的思想。

1. 关系优先框架将 N 转化为 Q：首先将 N 转化为 d^t，之后在 d^t 中寻找可以作为谓词 p 的短语，之后根据依赖关系以及一些启发式的规则向 p 的两侧添加结点(sub 与 obj)。但是这样做存在一个问题，关系优先框架不能发现 N 中存在的隐式关系。也就是说一个谓词 P（一个关系）没有直接出现在 N 中，那么这个关系，以及相关的结点都会被抛弃。实际表现为 Q 会缺少一个 triple pattern。
2. 结点优先框架解决了这个问题，结点优先框架首先在 d^t 中寻找可以做结点的词语。之后寻找结点之间的关系，如果 N 中没有显式的给出一个关系，但是实际存在两个结点，gAnswer 会根据统计信息来分配一个关系。

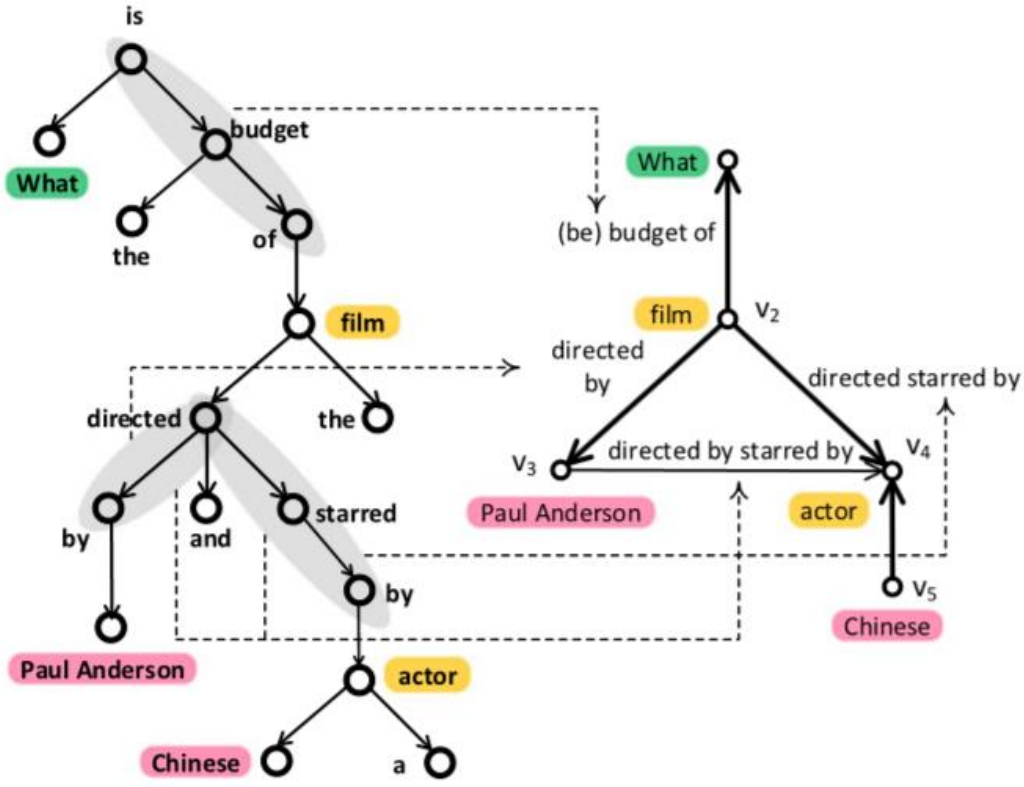
后续过程

DEFINITION 2. *(Semantic Query Graph) A semantic query graph(denoted as S^Q) is a graph, in which each vertex v_i is associated with an entity phrase or class phrase in the quesion sentence N;and each $\overrightarrow{v_i v_j}$ is associated with a relation phrase in the question sentence N.*

DEFINITION 3. *(Sparql Query Graph) A Sparql query graph(denoted as Q^G) is a graph, in which each vertex v_i is the corresponding entity in the RDF data for the v_i in the S^Q and each $\overrightarrow{v_i v_j}$ is the corresponding predicate in the RDF data for the $\overrightarrow{v_i v_j}$ in the S^Q.*

gAnswer 首先将 N 转化为 d^t 结构，之后再将 d^t 转化为 S^Q，最后将 S^Q 转化为 Q^G。

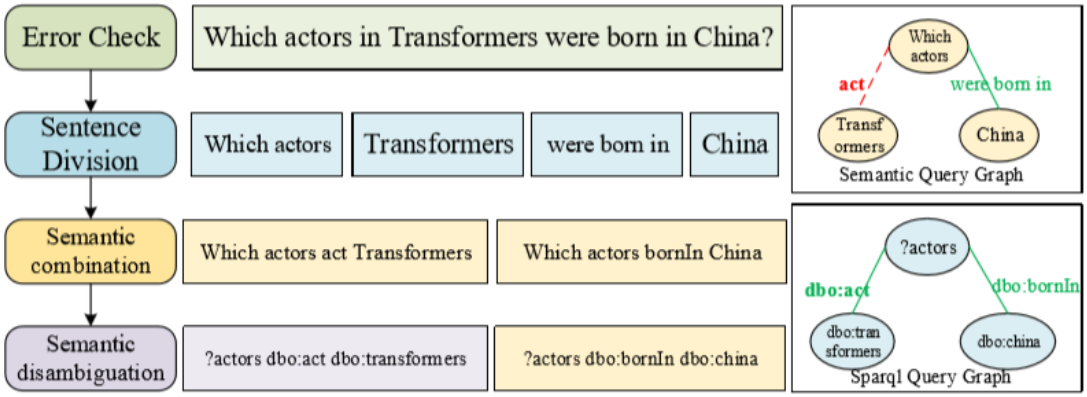
为了更好更快的从 d^t 中找出 S^Q，gAnswer 还提出了一个超图，可以将 d^t 大大简化。去除掉许多的无用信息。



上图中左图是一个 d^t,右图是一个超图。但是这一步存在比较大的

缺陷，在去除一些不重要结点的时候，依赖于提前收集和设计的统计信息。

有了超图以后就要寻找 S^Q 以及 Q^G，例子如下：



上图中 sementic query graph 中红色虚线 act 是一个隐式关系。因此 gAnswer 处理关键流程如下：

N	d^t	超图	S^Q	Q^G

本文关键创新点（针对第二三个挑战）：

整体处理架构

定义：

超句法分析树（SDT）：将 dt 进一步抽象可以得到。超句法分析树是创新点汇集区。在将句法分析树转化为超句法分析树的过程中，会有语义的增，删，以及修改操作。可以通过语义增操作实现语义的精准捕捉，通过语义删以及修改操作来提高查询效率。于此同时，寻找

超句法分析树之间的公共结构来进一步提高效率。

具体步骤如下：

第一步：$f(N) \rightarrow d^t$

使用斯坦福提供的 jar 包进行。

第二步：$f(d^t) \rightarrow d_t^s$

语义增：

例如给出这样一个问题，show the information about obama。针对这个问题，gAnswer 以及目前存在的所有的将 N 转化为 Q 的框架都无法解决。

问题在于，obama 可以在数据集中寻找到对应的实体。但是 information 不行。Information 是一个抽象概念，也就是说这个 information 类似于一个类，其中包含许多概念，例如在此处 information 可能包括（age, wife, father, country….），这一类问题目前是无法解决的。而这种类型的问题又很常见，例如，奥巴马的联系方式（电话,email ，wechat, 推特…）

解决思路：

首先定位问题。

（1）当一个 N 中只含有一个实体的情况下执行语义增。

（2）当 N 中含有多个实体，但是其中存在一部分结构只存在一个实体，而且这部分结构会修饰另外一个实体的时候，执行语义增。

根据统计信息	先补后排序
统计信息很好理解，通过提前进行数据分析，构造一个类似的<key,value>表。例如 key 为联系方式，value 为 tel ,email … 但是这一步存在的显著问题是如何构建这个统计信息表。	首先，查找出所有与这个实体相关的谓词集合 P^s，之后在这个集合中根据 N 中的其他辅助信息进行谓词的选择和重要度排序。（未解决）

语义改（等价代换）：

例如存在这样的一个问题，who is Obama's father's father？其生成的 sparql 查询为

Select ?name

{ ?x name Obama. ?x father ?y . ?y father ?name. }

但是事实上，这一步完全是可以简化的。因为 father's father 与 grandfather 是等价的。因此这个查询 Q 完全可以等价为以下这个查询。

Select ?name

{ ?x name Obama. ?x grandfather ?name. }

这一步，采用本体文件中的信息来进行等价代换。

语义删：

这一步最好理解，主要目的为只保留句法分析树中的核心实体，将无意义词语，以及对生成 Q 贡献不大的词语进行删除。

有数据集如下：

N	准确 Q
who is Obama's father?	Q1
Obama's father?	Q1
Obama father?	Q1
who is it or and …	null

将 N 中所有的单词向量化为 W_n。之后将 N 向量化为 V_n。V_n 为 N 中所有的 W_n 之和。

之后将 Q 向量化为 Q_v。

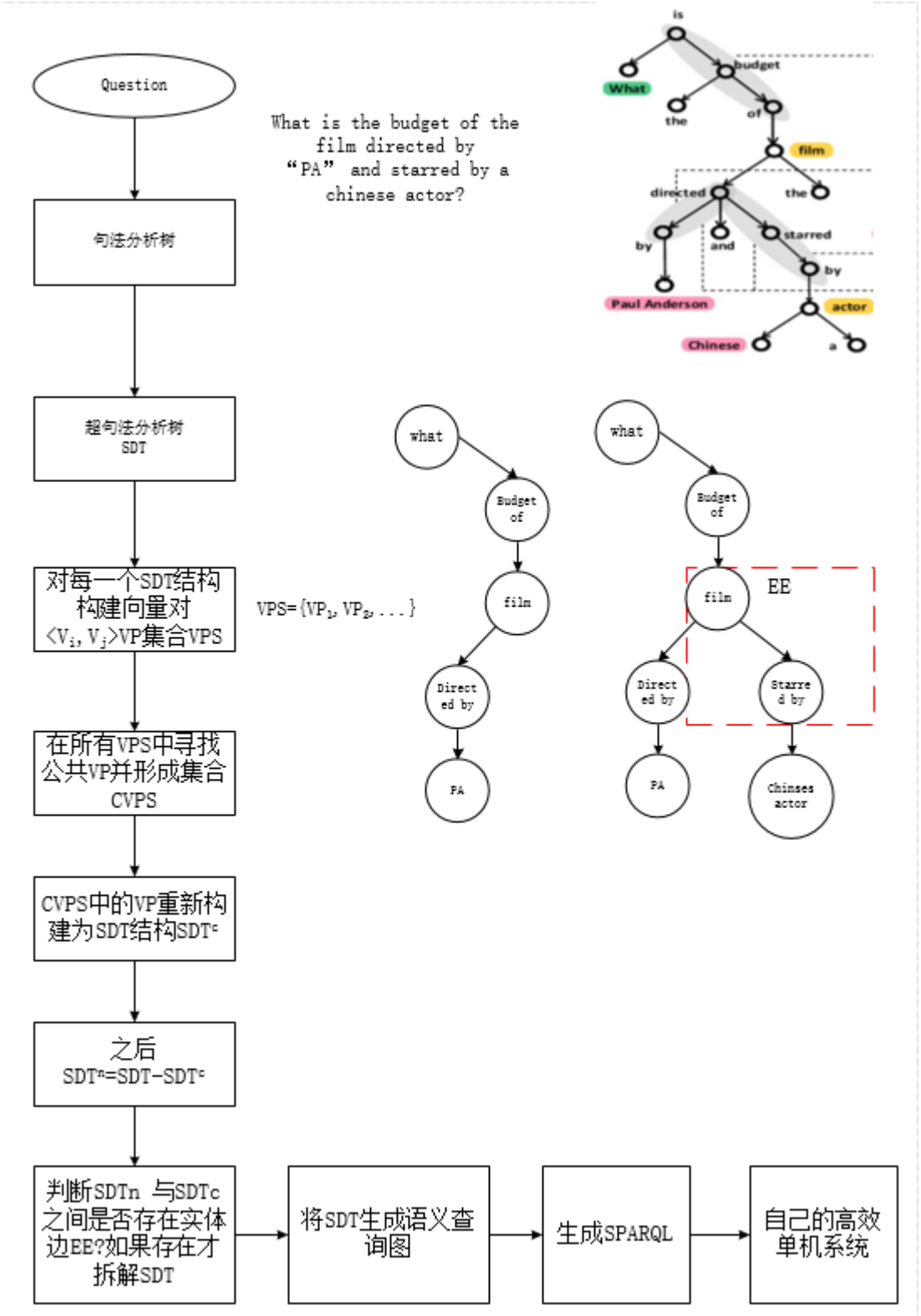
之后进行训练，并尽可能保持 $V_n = Q_v$.

整个数据集训练完毕后，得到了所有的 W_n。之后将 W_n 中所有维度值相加，其和为这个单词的核心度。因此可以根据核心度大小来选择删除一些单词，来提高效率。

这一步的依据是，一句话中不同的单词对 sparql 生成的影响力是不同的。

进一步：最好能给出一句话中不同位置的相同单词，重要性也是不同的。

第三步：寻找 SDT 之间的公共结构 SDT^C



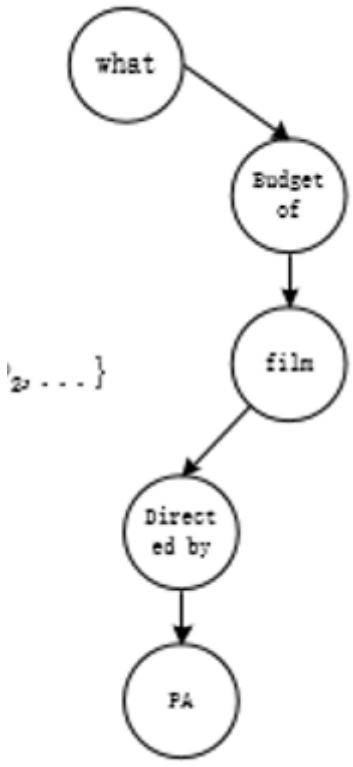
定义

二元组：由 SDT 之中的两个顶点值组成（V_i,V_j）,其中 V_i 与 V_j 之间拥有一条连接边，并且 V_i 指向 V_j。

EE：指的是同时存在于 SDT^C 与 SDT^n 中的公共结点，是一个实体。能在数据集中找到对应的结点。例如上图中的 film,

将 SDT 拆分为二元组的集合，如果两个树之间拥有共同结构，那么一定含有相同的二元组。

假设有 SDT 如下：



二元组如下：<what,budget of> , <budget of, film>, …

将二元组向量化之后，寻找所有超句法分析树中具有相同含义的二元组。

向量化的目的是假设有这么两个二元组，（dad,x）和(father,x)如

果不进行向量化，那么这两个二元组不同。如果向量化之后再判断，这两个二元组就变成相同的了。

之后找到了所有共同二元组。所有的共同二元组进行组装，可以组装出来一颗公共结构树。

接下来会借鉴 gAnswer 的内容。将公共超句法分析树与剩余的句法分析树生成 SPARQL。

如何快速执行公式 2

设计了一个新的单机版 RDF engine。主要是存储方式，以及 join 方式上有一些创新。创新不大。具体测试过后，能比较快的响应查询。

Lubm100 响应时间小于 0.5s。加载数据时间小于 90s。

测试为 windows 系统，8GB 内存。