# 基于知识图谱的批量Q/A系统

# 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Qc，而重复执行这些Qc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。

# 整体架构图以及思路

**N：**代表*question*

**SN：**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)

(1)

(2)

(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SN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SN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)：

(4)

有了公式4就可以先对SN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中的同步动作。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。

1. 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Tc。之后分别将（Tc）与(T - Tc)转化为Qc和Q-。

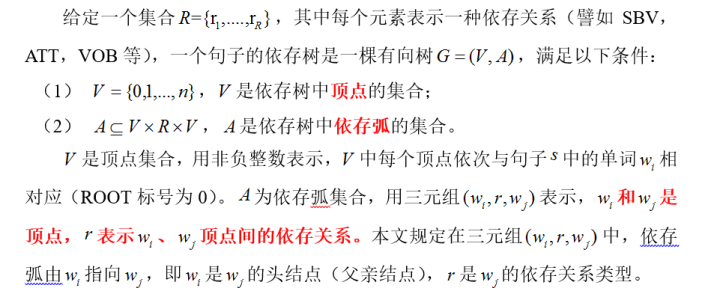
假设Qc首先被生成，Qc就先执行公式2，获取结果Rc，之后Q-生成，再对Q-执行公式2，生成R-。最后执行Rc⋈R-就可以得到最后的结果。

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。

# 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。

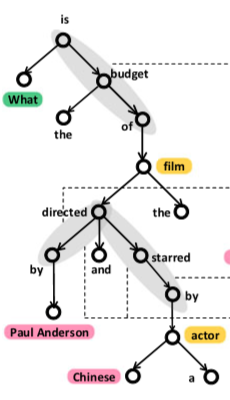
**句法依存树(denoted as *dt* )的定义：**



例子：

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

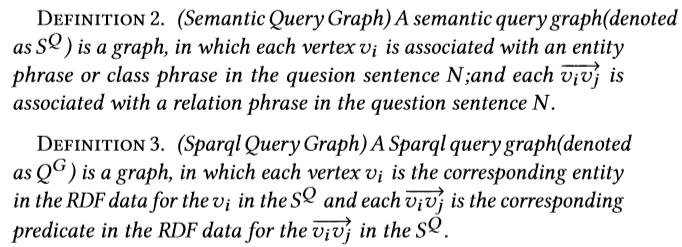
其简化版句法依存树如下，这个句法分析树只保留了结构，没有显示每个词的词性，以及词语之间的关系。



接下来简短的描述两种算法的思想。

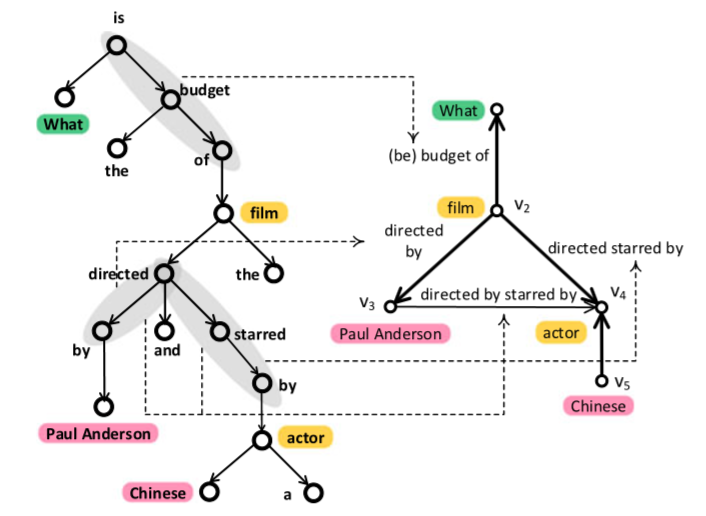
1. 关系优先框架将N转化为Q：首先将N转化为*dt*，之后在*dt中寻找可以作为谓词p的短语，*之后根据依赖关系以及一些启发式的规则向p的两侧添加结点(sub与obj)。但是这样做存在一个问题，关系优先框架不能发现N中存在的隐式关系。也就是说一个谓词P（一个关系）没有直接出现在N中，那么这个关系，以及相关的结点都会被抛弃。实际表现为Q会缺少一个triple pattern。
2. 结点优先框架解决了这个问题，结点优先框架首先在*dt*中寻找可以做结点的词语。之后寻找结点之间的关系，如果N中没有显式的给出一个关系，但是实际存在两个结点，gAnswer会根据统计信息来分配一个关系。

## 后续过程



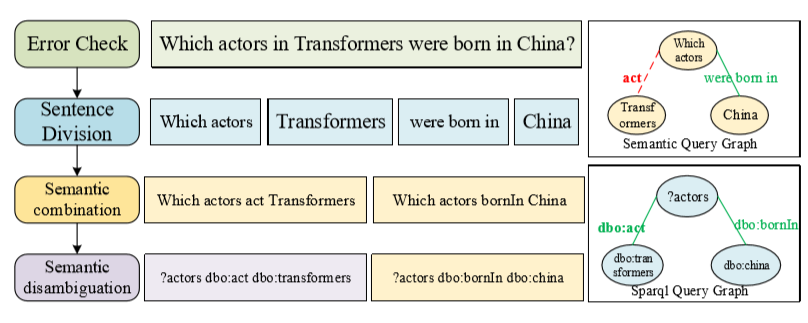
gAnswer首先将N转化为dt结构，之后再将dt转化为SQ，最后将SQ转化为QG。

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



上图中左图是一个dt,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SQ以及QG，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 | dt | 超图 | SQ | QG |

# 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) -> dt**

使用斯坦福提供的jar包进行。

### 第二步：f(dt) -> dts

### 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Ps，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Wn。之后将N向量化为Vn。Vn为N中所有的Wn之和。

之后将Q向量化为Qv。

之后进行训练，并尽可能保持Vn = Qv.

整个数据集训练完毕后，得到了所有的Wn。之后将Wn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。

# 公共结构以及分组查询(parallel Or distributed)

**定义1：**

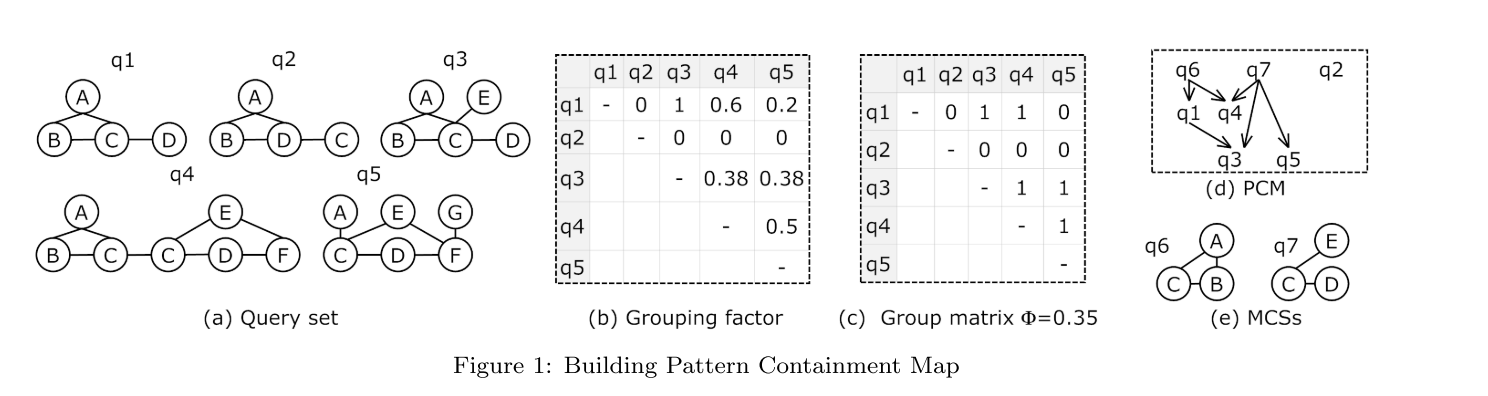
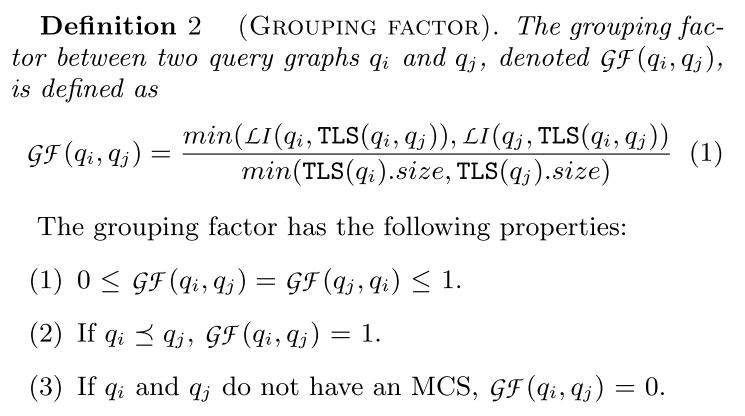
**(Tri-Vertex Label Sequence)** (Tri-Vertex Label Sequence). Given a pair of connected edges (vi, vj) and (vj, vk) of a super dependency tree TS, assuming L(vi) ≤ L(vk), we call the label sequence L(vi)- L(vj)-L(vk) a Tri-Vertex Label Sequence (TLS), and (vi, vj, vk) an instance of the TLS in TS. 

Figure 1. Super dependency tree set

Figure 1. TLS (TS3, TS4) ={(A-B-C), (A-C-B), (B-A-C), (D- C-E)}. We use L1(TS3, TLS(TSi, TSj)) to denote the number of instances in the largest instance subTree of TS3 corresponding to the TLSs in TLS(TSi, TSj). For example, for TS3, and TS4 in Figure 1, L1(TS3, TLS(TS3, TS4)) =4 and L1(TS4, TLS(TS3, TS4))=3。

之后根据如下定义计算grouping factor并根据阈值形成Fiuger1 c。



最终形成Figure 1中的d。

在这里PCM中的每一个箭头都代表两个TS之间存在公共结构。

# 3.分组执行

从这个地方开始不同于论文中的工作，我的设想是到这一步开始，将PCM中的超句法依存树进行分组。

即便是单机的系统，其并行度一般也不为1。也就是说，即便是只能在单机上执行的程序我们仍然可以根据硬件条件（CPU数量与核数），写出多线程或者多进程代码，因此程序可以并行执行。例如，对同一份RDF数据，同时执行SPARQL查询操作是没有问题的。

因此这引出一个新的问题。如何将查询进行分组，使其对应硬件的并行度。例如执行SPARQL查询时可能有这样的问题。有n个查询（不考虑公共结构），电脑的并行度是m，那么应该如何将这n个查询尽可能平均的分为m组非常重要。当然是尽可能的平均分最好。于是这个问题一定会引出下面这个问题，估计查询代价。

## 3.1 查询代价(时间)

假设有估计查询代价的函数cost(TS)，通过这个函数我们可以估计一个由句法依存树查询TS生成的SPARQL查询Q的代价。当然最好有两个代价函数:

(1)

(2)

其中公式1可以给出TS生成的SPARQL查询Q的代价，公式2可以给出由 生成的Qi和Qj公共查询Qc的代价。于是如果我们设计了多查询优化，公共查询只执行一次，于是就会有如下公式：

（3）

拥有了公式3，就可以计算在有公共结构的情况下，一组超句法分析树生成的SPARQL查询的代价。

目前已经有不少研究工作给出了使用机器学习的方法来预测SPARQL查询的代价的方法，例如14年的论文< A Machine Learning Approach to SPARQL Query Performance Prediction >以及18年WWW的一篇论文都给出了预测SPARQL查询代价的方法。

相比较于单纯的SPARQL查询给出的特征信息，事实上句法分析树中给出的特征信息更多。因此希望可以找到一个比较中肯的方法（机器学习算法）来实现预测由超句法分析树生成的SPARQL查询的代价。

## 3.2 PCM中句法分析树分组

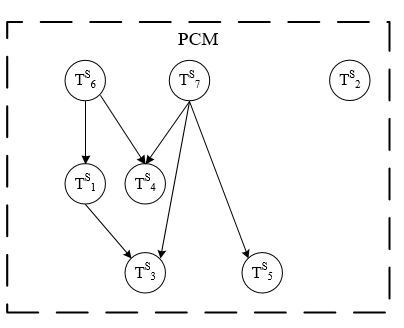


Figure 2 PCM

如图2所示，假设超句法分析树之间可以形成这样的连接关系。

**定义：**

**入度**：指向PCM中某一结点TSi的边数量。

**出度**：PCM中某一结点TSi指向其他结点的边数量。

**度**：入度+出度

不难看到，PCM中的所有结点可以分为以下三种情况

* 度为0
* 度为1
* 度>=2

分组的时候，对于度为0和1的结点都很好解决。度为0 的结点例如TS2不和任何其他结点有公共结构，因此自己就是一组。度为1的结点，例如TS5就一定是和指向它的结点TS4绑定在一组。

而针对度大于等于2的结点TSi就比较复杂。TSi可以与所有指向它的结点组合在一组，也可以与它指向的结点组成一组。为了更好的负载均衡。设计如下启发式的算法进行分组。

**算法如下：**

**输入**：PCM ，电脑并行度N

**输出**：将PCM中的所有超句法分析树分为N组

算法分为三种情况：

***PCM中度为1的结点数量>=N***

1. 挑出所有度为1的结点，分为N组。
2. 将所有与度为1的结点相连的结点加入对应组
3. 计算每组的查询代价
4. 将度为大于等于2的结点尝试加入有边与其相连的其他结点所属分组，（要加入查询代价最小的一组）
5. 最后分配度为0的结点。尽可能保持不同组之间代价基本相似。

**PCM中度为1的结点的数量>N/2且<N**

1. 按照度为1的结点的数量m分为m组
2. 将所有与度为1的结点相连的结点加入对应组

由于实际上服务器的并行度为N，而m<N，因此还剩下N-m分组可以进行计算。

1. 因此此步选择将度为0的结点平均分入这N-m个分组中。（但是要保证这N-m个分组中，最大的那一组代价不能超过那m个分组中的最大代价，如果开始超出，则将该超句法分析树分配至前m个分组中代价最小的一组）
2. 处理度大于等于2的结点。每次都选择当前m组中代价最小的那一组加入。（未必要是与该结点连的结点所属的组）

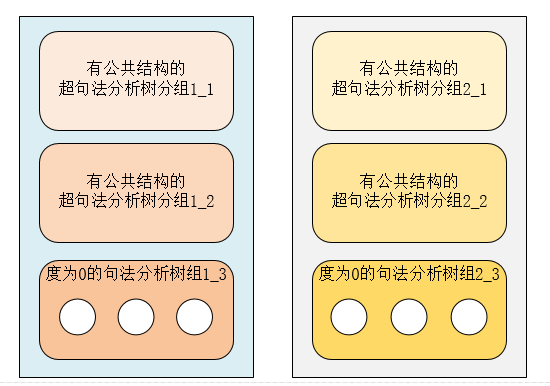
**PCM中度为1的结点数量小于N/2时**

未解决。

N/2是一个设定的参数，可以是N/2,也可以是N/3等等

## 组查询执行顺序问题

经过上面对PCM中句法分析树的分组。给出如下一个例子：



假设计算机的并行度为2，并生成了如上图的两个分组。在超句法分析树生成SPARQL查询Q的部分是没有问题的。这只涉及到计算。计算过程中对存储的压力依赖比较小。

但是当所有的超句法分析树都生成为SPARQL之后，要执行SPARQL查询问题就来了。上图中每个组对内存的耗费情况是不同的。 例如每一组从开始执行到执行完毕这段时间内耗费内存情况如下：

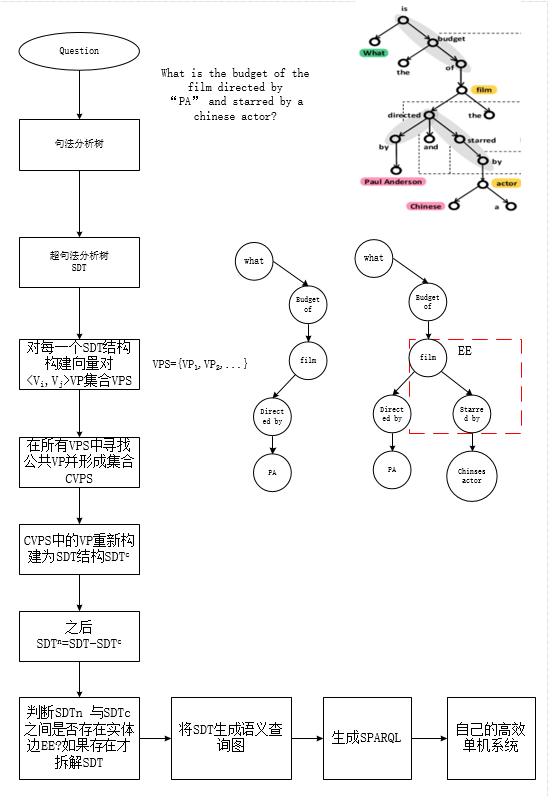
|  |  |  |
| --- | --- | --- |
| 执行顺序 | 组1 | 组2 |
| 1 | 500MB | 500MB |
| 2 | 100MB | 200MB |
| 3 | 3GB | 2GB |

如果只是单纯的要按照上表的执行顺序来执行。如果计算机的内存只有4GB，那么很显然。当执行第三步的时候麻烦就大了。由于内存受限。无法运行。计算机崩溃。但这本来是可以避免的。

**因此在此步应该有两步重要内容需要做。**

1. 如何使用机器学习算法来估测sparql查询占用内存的大小（可以使用每个triple pattern对应的数据所占存储空间大小之和）
2. 设计合理的組执行顺序算法。

### 第三步：寻找SDT之间的公共结构SDTC



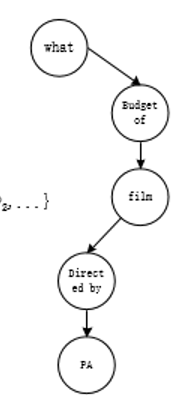
**定义**

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

**EE：**指的是同时存在于SDTC与SDTn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。