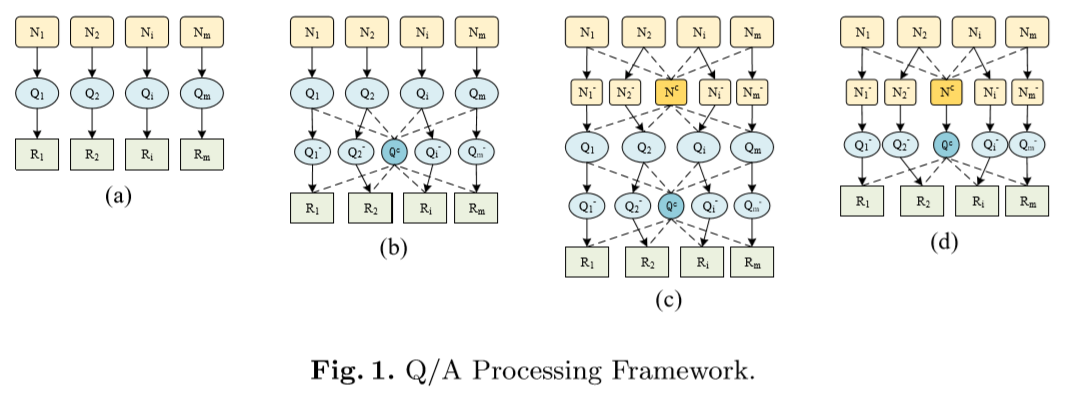
最大的贡献是提出了如下批量Q/A处理框架，但是还要回答如下几个问题。最大的贡献来源于两点：

1. 单个question（速度基本一致，但是能处理更多情况）
2. 批量question处理方法
   1. 精度更高
   2. 效率更快
3. 给出并行化处理方法（扩展工作中给出分布式设计方法）



# Nc的含义与获取方法

Nc指的是公共语义。要想找到两句话中的公共语义，有两个挑战要解决。

**其一**：判断两句话是否是语义相似的。

**其二**：如何在两个Question中找到公共语义部分。

为方便介绍，我们使用N来代表a natual language question，用wi代表组成N的单词，使用Q代表由N生成的SPARQL查询。

## 判断语义相似度

目前NLP中判断两句话之间是否相似的方法有很多，我们选择使用google提出的doc2vec模型来判断N之间的相似性。

Doc2vec是word2vec的进阶版本。能很好的判断两句话或者两个文本块之间的相似度。

## 在N之间找出相似的语义段

### 穷举方法寻找

|  |  |  |  |
| --- | --- | --- | --- |
|  | N1 | N2 | 相似度 |
| 1 | who is obama’s father? | who is Obama’s dad? | 0.99 |
| 2 | who is obama’s | who is Obama’s | 1 |
| 3 | is obama’s father? | is Obama’s dad? | 0.99 |
| 4 | who is | who is | 1 |
| 5 | is obama’s | is Obama’s | 1 |
| 6 | obama’s father? | Obama’s dad? | 0.98 |

可以看到任何一个包含m个单词的问题N都可以被分为一系列的子问题，子问题数量由如下公式定义：

事实上有N1产生的所有子问题与N2产生的所有子问题之间都要进行一次比对。时间复杂度为O(n2)。

### 启发式寻找方法

通常来说，一个N中wi的数量比较少，因此上面的穷举方法已经足够了。但是也不能避免一些用户会输入比较长的N。因此上面的穷举方法就不够高效了。由此我们给出如下启发式方法。

例子：

我是一个研究生，我的名字是张三，我就读于天津大学。

我是一个研究生，我的名字是李四，我的性别是男，我就读于天津大学。

**第一步：**挑选出可能成为SPARQL中实体（主语、谓语）或者谓词的词语。

我 研究生，我 名字 张三，我 就读于 天津大学。

我 研究生，我 名字 李四，我 性别 男，我 就读于 天津大学。

**第二步**：挑选出两个N之间所有的满足相似性的单词。(word2vec)

我 研究生，我 名字 张三，我 就读于 天津大学。 N1

我 研究生，我 名字 李四，我 性别 男，我 就读于 天津大学。 N2

**第三步：**具体的算法：

找出较短句子N1中所有连续的相似单词（绿色单词连接在一起的部分）。

## 边界处理

主要处理的是红色与绿色单词相邻的区域。

**标红的部分是一个单词：**

这个单词单词的前后必然存在一部分绿色内容。（从句法解析树上看）如果就近的绿色内容中包含一个可以做谓语的单词，那么这个单词很可能将来会是一个triple pattern的主语或者宾语。此时这个单词会被保存起来作为以后的一个筛选条件。

如果就近的绿色内容是一个可以作为主语或者宾语的单词，那就说明存在隐式关系需要挖掘。因此这个距离标红单词最近的标绿单词被取出来不再作为生成公共结构的一部分

**标红的部分是一个短句：**

就当做一个子问题处理。将其生成为triple pattern。后面在进一步处理。

# 由Nc生成Qc的优点

上节介绍过Nc的含义以及如何找到Nc。这节中主要介绍由Nc生成Qc的优点。

**定义[不完全覆盖问题]**：指的是SPARQL不能完全覆盖Nc表达的语义。

我们知道要想完美的使用SPARQL来代替N的语义是很困难的。因此经常会出现不完全覆盖问题。给一些典型的例子。

**例子：**

N1: 原子能的

N2:原子能是什么与原子的应用

N1问题原本是想问，原子能的应用。但是由于各种可能的原因会出现如下错误。因此N1可能会变成：

原子能的

原子能的英勇…

当前的N1,只能生成如下SPARQL查询：

{

?x name 原子能

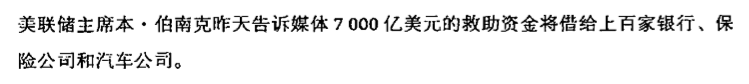
}

这个SPARQL查询只覆盖了原来的N1中的原子能这个内容。由于无法识别“的应”是什么只能选择抛弃这部分内容。

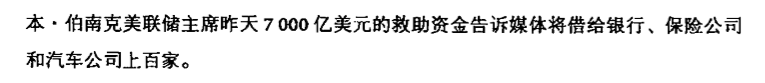
由于直接在SPARQL中寻找公共结构的时候，这部分信息已经损失掉了。因此我们是无从得知是否存在不完全覆盖问题。因此只能将错就错。而选择使用Nc生成Qc就可以一定程度避免这个问题。

## NLP中判断句子是否有意义

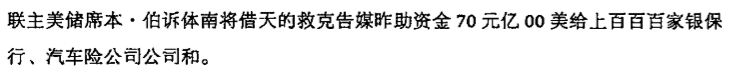
（1）



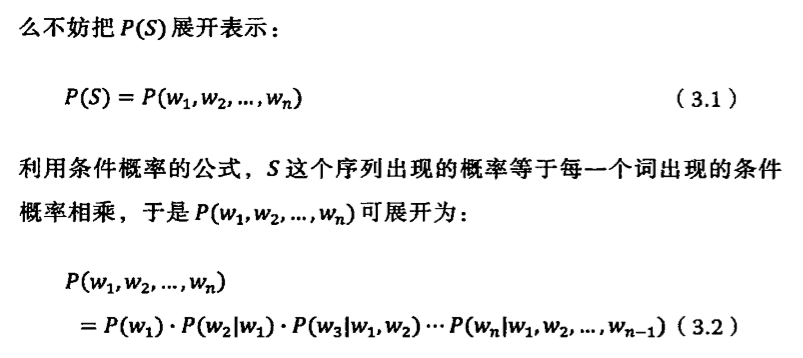
（2）

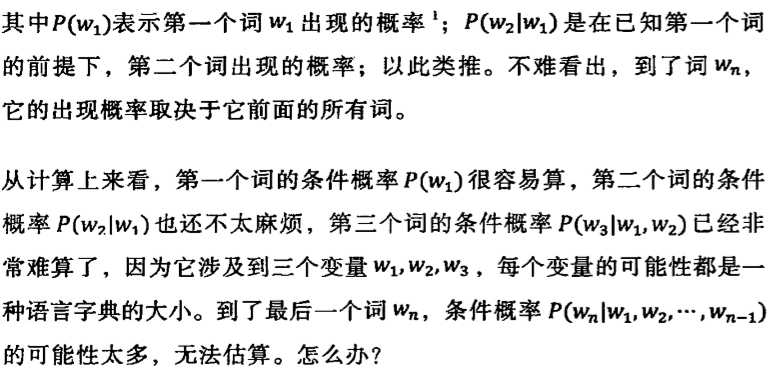


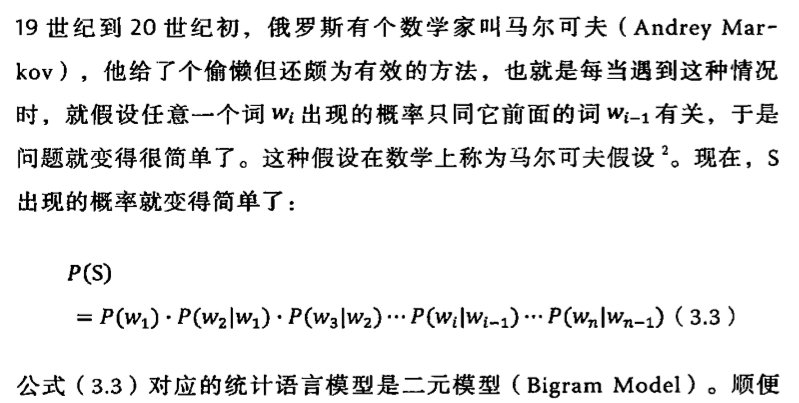
（3）

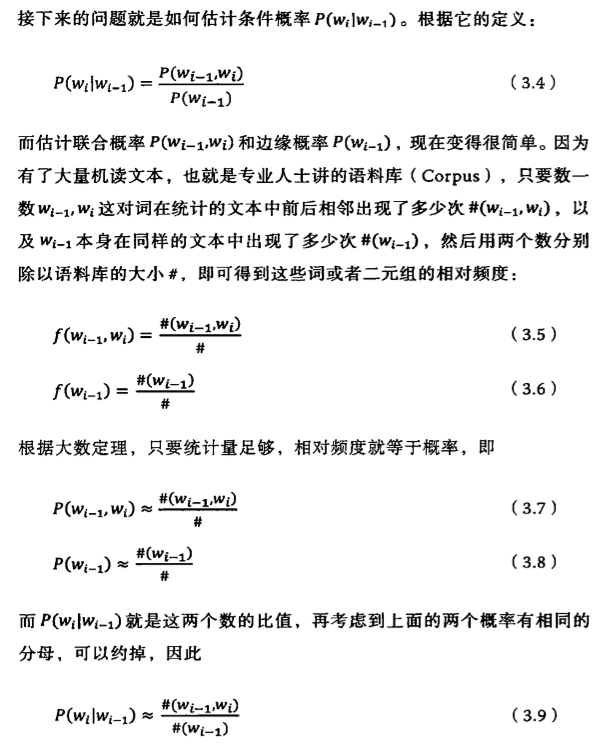


我们用S代表一个句子。









现在把S看做一个完整的sparql查询。而Wi就是SPARQL中的triple pattern。于是我们可以通过一样的计算方法，使用隐马尔可夫模型来挑选出在具有的当前triple pattern的基础上，哪个triple pattern是最有可能被选出来的。

这个最可能被选出来的triple pattern就是我们选择出用来覆盖N语义的。（语义增）

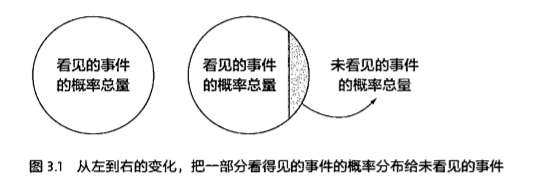
但是由于训练数据量的不足够，可能会出现这样的问题。如果有几千个SPARQL查询用来训练，其中几百个与原子能相关的SPARQL查询反映的语义都是“原子能的应用”。还有几十个SPARQL查询表达的语义是“原子能的制作”。这种情况下我们基本可以人为，“计算机的”这个问题，很可能用户想问的是“原子能的应用”。

但是如果总共只5个与原子能相关的SPARQL查询。其中4个表达的语义是“原子能的应用”，而只有1个SPARQL查询的语义是“原子能的制作”。那么直接说用户的这个问题是原子能的应用的概率是80%显然是不太好的。

因此我们需要解决当样本数量并不是太大的时候，应该怎样进行概率估计这个问题。

**古德图灵估计：**

对于未知的时间，我们不能认为它发生的概率为0.因此我们要从概率的总量中分很小的比例给予这些没看见的事件。这样一来看的见的那些事件的概率总和就小于1了。具体少多少根据原则“越是不可信的统计折扣越多来进行”。

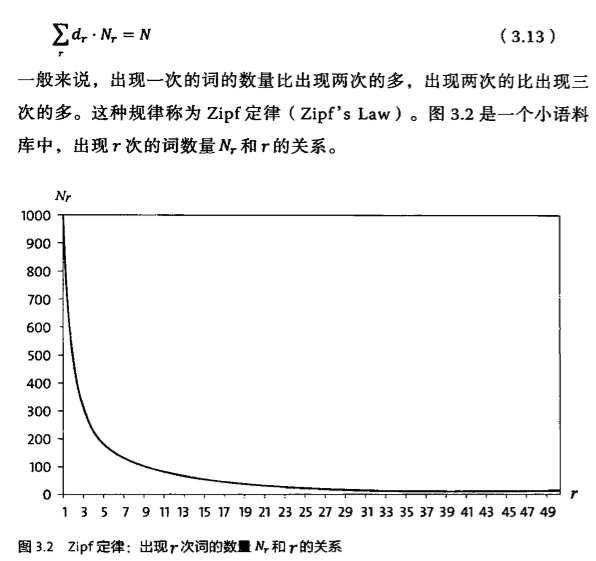


假设SPARQL语料库中，语料库中SPARQL查询的数量为N。

出现r次的SPARQL查询有Nr个，有如下公式：

一般来说，我们直接使用了r/N来表示这个SPARQL的概率了。但是当r比较小的时候，统计信息可能是不可靠的。因此出现r次的那些SPARQL查询在计算他们的概率时要稍微小一些，使用dr来计算。

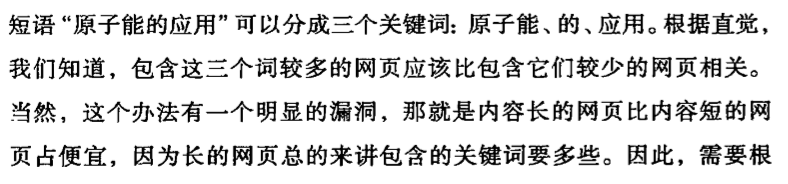


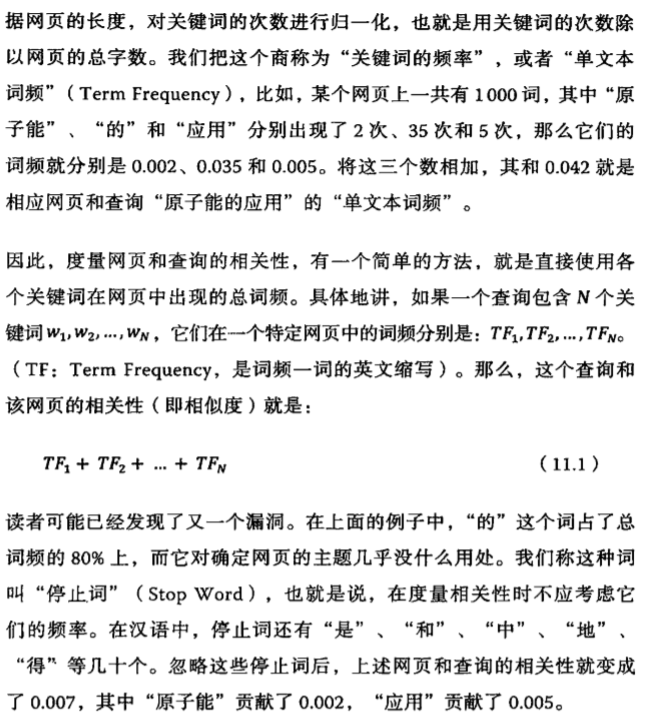


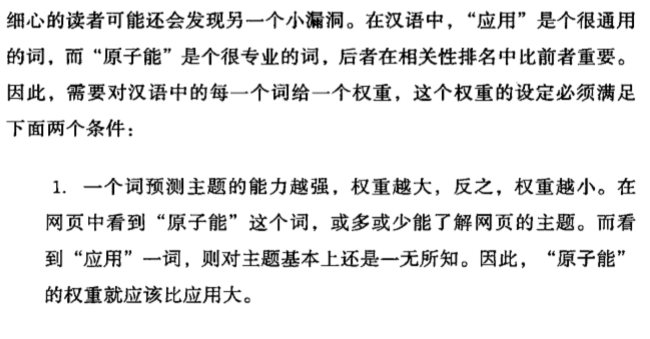
因此使用这种方法做到了对概率的平滑处理。

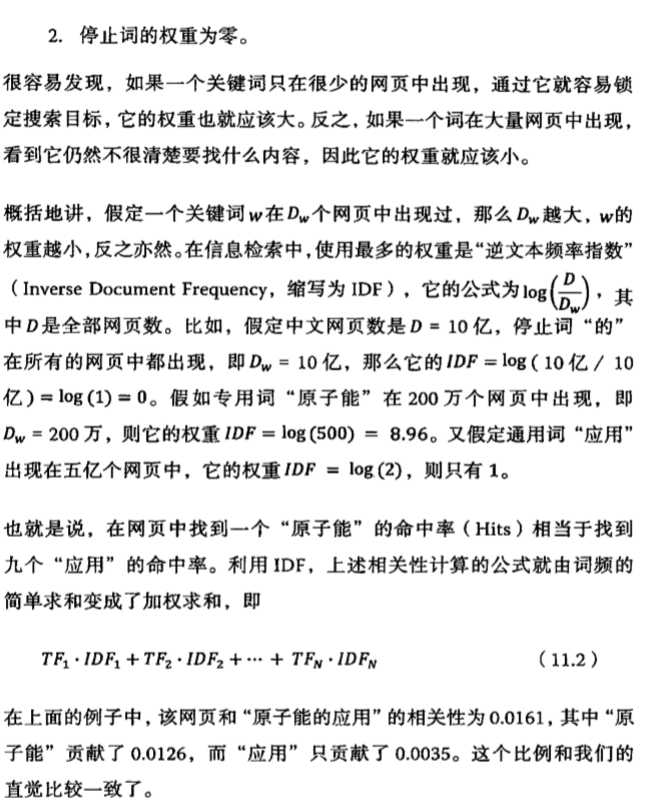
# SPARQL中triple pattern的核心度度量

Sparql中每个triple pattern对SPARQL的限制性是不同的。有的限制性更大，有的限制性更小。引入NLP中的例子：









之后按照这个例子，我们可以对SPARQL中triple pattern的核心度定义如下：

# 公共结构以及分组查询(parallel Or distributed)

**定义1：**

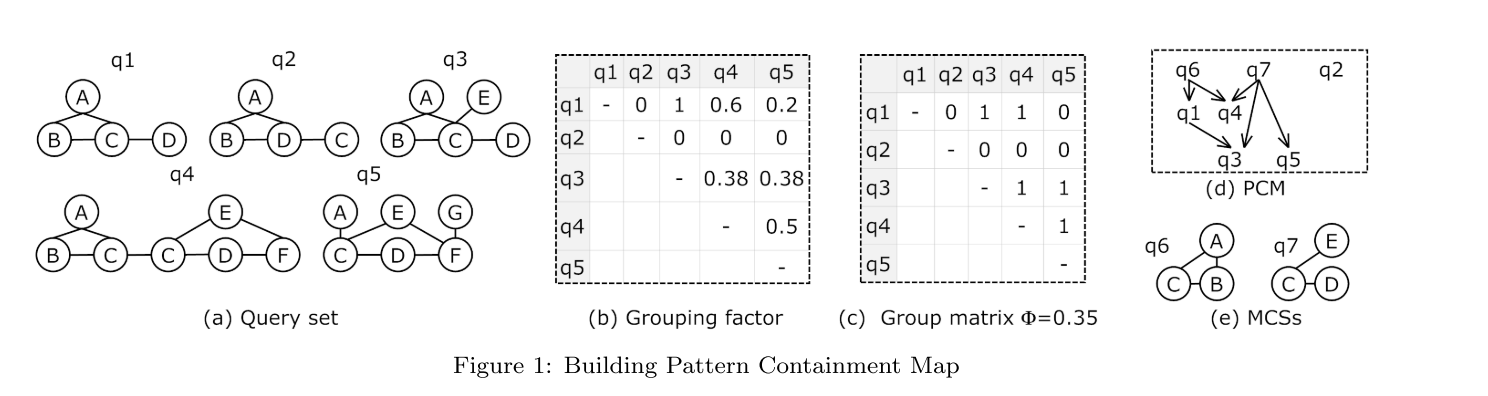
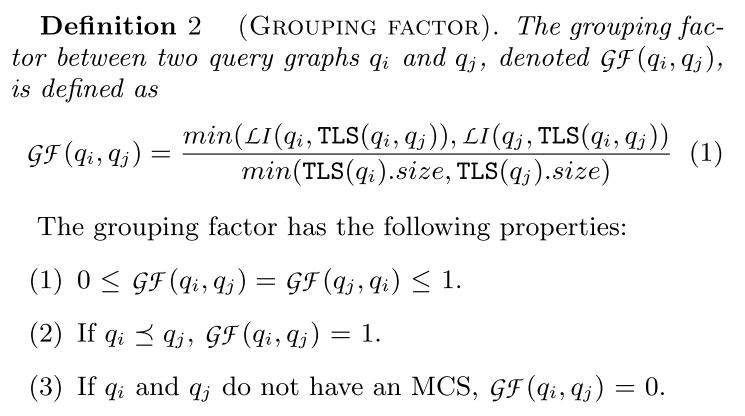
**(Tri-Vertex Label Sequence)** (Tri-Vertex Label Sequence). Given a pair of connected edges (vi, vj) and (vj, vk) of a super dependency tree TS, assuming L(vi) ≤ L(vk), we call the label sequence L(vi)- L(vj)-L(vk) a Tri-Vertex Label Sequence (TLS), and (vi, vj, vk) an instance of the TLS in TS. 

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

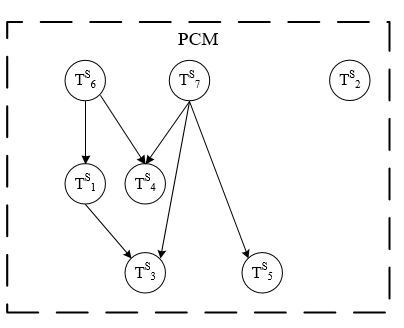


Figure 2 PCM

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）

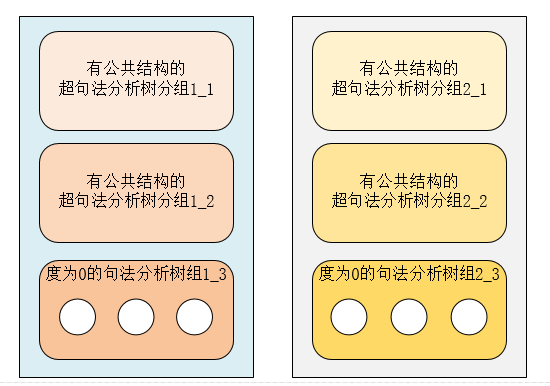
**PCM中度为1的结点数量小于N/2时**

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。