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

a) 精度更高

b) 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）

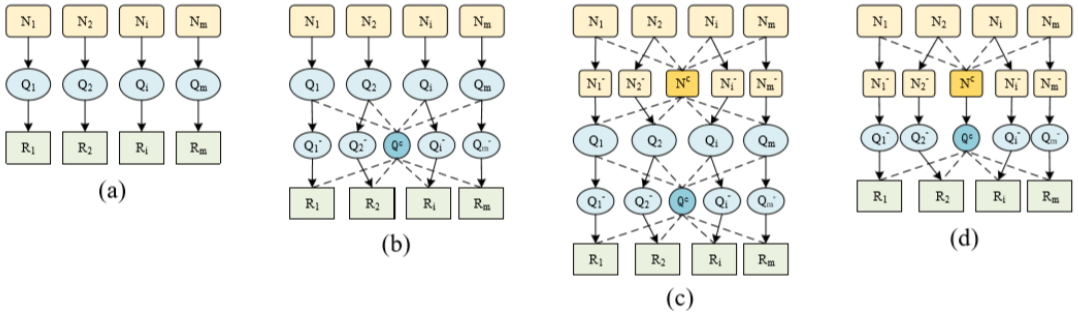


Fig. 1. Q/A Processing Framework.

1. 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。

1.1. 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。

1.2. 在 N 之间找出相似的语义段

1.2.1. 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：

$$\text{number} = \sum_{i=1}^{m-1} i$$

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 $O(n^2)$。

1.2.2. 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）。

1.3. 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。

2. 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。

2.1. NLP 中判断句子是否有意义

（1）

美联储主席本·伯南克昨天告诉媒体 7 000 亿美元的救助资金将借给上百家银行、保险公司和汽车公司。

（2）

本·伯南克美联储主席昨天 7 000 亿美元的救助资金告诉媒体将借给银行、保险公司和汽车公司上百家。

（3）

联主美储席本·伯诉体南将借天的救克告媒昨助资金 70 元亿 00 美给上百百百家银保行、汽车险公司公司和。

我们用 S 代表一个句子。

么不妨把 $P(S)$ 展开表示：

$$P(S) = P(w_1, w_2, \ldots, w_n) \tag{3.1}$$

利用条件概率的公式，S 这个序列出现的概率等于每一个词出现的条件概率相乘，于是 $P(w_1, w_2, \ldots, w_n)$ 可展开为：

$$\begin{aligned} &P(w_1, w_2, \ldots, w_n) \\ &= P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \ldots, w_{n-1}) \end{aligned} \tag{3.2}$$

其中 $P(w_1)$ 表示第一个词 w_1 出现的概率[1]；$P(w_2|w_1)$ 是在已知第一个词的前提下，第二个词出现的概率；以此类推。不难看出，到了词 w_n，它的出现概率取决于它前面的所有词。

从计算上来看，第一个词的条件概率 $P(w_1)$ 很容易算，第二个词的条件概率 $P(w_2|w_1)$ 也还不太麻烦，第三个词的条件概率 $P(w_3|w_1, w_2)$ 已经非常难算了，因为它涉及到三个变量 w_1, w_2, w_3，每个变量的可能性都是一种语言字典的大小。到了最后一个词 w_n，条件概率 $P(w_n|w_1, w_2, \cdots, w_{n-1})$ 的可能性太多，无法估算。怎么办？

19 世纪到 20 世纪初，俄罗斯有个数学家叫马尔可夫（Andrey Markov），他给了个偷懒但还颇为有效的方法，也就是每当遇到这种情况时，就假设任意一个词 w_i 出现的概率只同它前面的词 w_{i-1} 有关，于是问题就变得很简单了。这种假设在数学上称为马尔可夫假设[2]。现在，S 出现的概率就变得简单了：

$$\begin{aligned} &P(S) \\ &= P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_i|w_{i-1}) \cdots P(w_n|w_{n-1}) \end{aligned} \tag{3.3}$$

公式（3.3）对应的统计语言模型是二元模型（Bigram Model）。顺便

接下来的问题就是如何估计条件概率 $P(w_i|w_{i-1})$。根据它的定义：

$$P(w_i|w_{i-1}) = \frac{P(w_{i-1},w_i)}{P(w_{i-1})} \quad (3.4)$$

而估计联合概率 $P(w_{i-1},w_i)$ 和边缘概率 $P(w_{i-1})$，现在变得很简单。因为有了大量机读文本，也就是专业人士讲的语料库（Corpus），只要数一数 w_{i-1},w_i 这对词在统计的文本中前后相邻出现了多少次 $\#(w_{i-1},w_i)$，以及 w_{i-1} 本身在同样的文本中出现了多少次 $\#(w_{i-1})$，然后用两个数分别除以语料库的大小 $\#$，即可得到这些词或者二元组的相对频度：

$$f(w_{i-1},w_i) = \frac{\#(w_{i-1},w_i)}{\#} \quad (3.5)$$

$$f(w_{i-1}) = \frac{\#(w_{i-1})}{\#} \quad (3.6)$$

根据大数定理，只要统计量足够，相对频度就等于概率，即

$$P(w_{i-1},w_i) \approx \frac{\#(w_{i-1},w_i)}{\#} \quad (3.7)$$

$$P(w_{i-1}) \approx \frac{\#(w_{i-1})}{\#} \quad (3.8)$$

而 $P(w_i|w_{i-1})$ 就是这两个数的比值，再考虑到上面的两个概率有相同的分母，可以约掉，因此

$$P(w_i|w_{i-1}) \approx \frac{\#(w_{i-1},w_i)}{\#(w_{i-1})} \quad (3.9)$$

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”。

图 3.1　从左到右的变化，把一部分看得见的事件的概率分布给未看见的事件

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：

$$N = \sum_{r=1}^{无穷} r * Nr$$

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。

$$d_r = (r+1) \cdot N_{r+1}/N_r \qquad (3.12)$$

$$\sum_r d_r \cdot N_r = N \qquad (3.13)$$

一般来说，出现一次的词的数量比出现两次的多，出现两次的比出现三次的多。这种规律称为 Zipf 定律（Zipf's Law）。图 3.2 是一个小语料库中，出现 r 次的词数量 N_r 和 r 的关系。

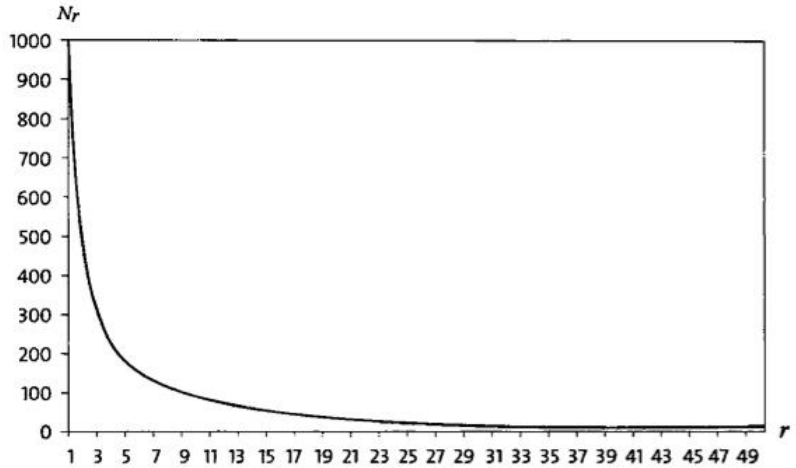


图 3.2 Zipf 定律：出现 r 次词的数量 N_r 和 r 的关系

因此使用这种方法做到了对概率的平滑处理。

3. 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：

短语“原子能的应用”可以分成三个关键词：原子能、的、应用。根据直觉，我们知道，包含这三个词较多的网页应该比包含它们较少的网页相关。当然，这个办法有一个明显的漏洞，那就是内容长的网页比内容短的网页占便宜，因为长的网页总的来讲包含的关键词要多些。因此，需要根据网页的长度，对关键词的次数进行归一化，也就是用关键词的次数除以网页的总字数。我们把这个商称为“关键词的频率”，或者“单文本词频”（Term Frequency），比如，某个网页上一共有 1000 词，其中“原子能”、“的”和“应用”分别出现了 2 次、35 次和 5 次，那么它们的词频就分别是 0.002、0.035 和 0.005。将这三个数相加，其和 0.042 就是相应网页和查询“原子能的应用”的“单文本词频”。

因此，度量网页和查询的相关性，有一个简单的方法，就是直接使用各个关键词在网页中出现的总词频。具体地讲，如果一个查询包含 N 个关键词 $w_1, w_2, \ldots, w_N$，它们在一个特定网页中的词频分别是：$TF_1, TF_2, \ldots, TF_N$。（TF：Term Frequency，是词频—词的英文缩写）。那么，这个查询和该网页的相关性（即相似度）就是：

$$TF_1 + TF_2 + \ldots + TF_N \qquad (11.1)$$

读者可能已经发现了又一个漏洞。在上面的例子中，“的”这个词占了总词频的 80% 上，而它对确定网页的主题几乎没什么用处。我们称这种词叫“停止词”（Stop Word），也就是说，在度量相关性时不应考虑它们的频率。在汉语中，停止词还有“是”、“和”、“中”、“地”、“得”等几十个。忽略这些停止词后，上述网页和查询的相关性就变成了 0.007，其中“原子能”贡献了 0.002，“应用”贡献了 0.005。

细心的读者可能还会发现另一个小漏洞。在汉语中，“应用”是个很通用的词，而“原子能”是个很专业的词，后者在相关性排名中比前者重要。因此，需要对汉语中的每一个词给一个权重，这个权重的设定必须满足下面两个条件：

1. 一个词预测主题的能力越强，权重越大，反之，权重越小。在网页中看到“原子能”这个词，或多或少能了解网页的主题。而看到“应用”一词，则对主题基本上还是一无所知。因此，“原子能”的权重就应该比应用大。

2. 停止词的权重为零。

很容易发现，如果一个关键词只在很少的网页中出现，通过它就容易锁定搜索目标，它的权重也就应该大。反之，如果一个词在大量网页中出现，看到它仍然不很清楚要找什么内容，因此它的权重就应该小。

概括地讲，假定一个关键词w在D_w个网页中出现过，那么D_w越大，w的权重越小，反之亦然。在信息检索中，使用最多的权重是“逆文本频率指数”（Inverse Document Frequency，缩写为IDF），它的公式为$\log\left(\frac{D}{D_w}\right)$，其中$D$是全部网页数。比如，假定中文网页数是$D$ = 10亿，停止词“的”在所有的网页中都出现，即D_w = 10亿，那么它的$IDF = \log$（10亿 / 10亿）$= \log(1) = 0$。假如专用词“原子能”在200万个网页中出现，即D_w = 200万，则它的权重$IDF = \log(500) = 8.96$。又假定通用词“应用”出现在五亿个网页中，它的权重$IDF = \log(2)$，则只有1。

也就是说，在网页中找到一个“原子能”的命中率（Hits）相当于找到九个“应用”的命中率。利用IDF，上述相关性计算的公式就由词频的简单求和变成了加权求和，即

$$TF_1 \cdot IDF_1 + TF_2 \cdot IDF_2 + \cdots + TF_N \cdot IDF_N \qquad (11.2)$$

在上面的例子中，该网页和“原子能的应用”的相关性为0.0161，其中“原子能”贡献了0.0126，而“应用”只贡献了0.0035。这个比例和我们的直觉比较一致了。

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：

$$C\quad ore = \left(\frac{\text{一个 SPARQL 中含有这个 triple 的数量}}{\text{SPARQL 中的 triple 数量}}\right) * \left(\frac{\text{所有 SPARQL 数量}}{\text{含有这个 triple 的 SPARQL 数量}}\right)$$

4. 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 T^S, 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 T^S.

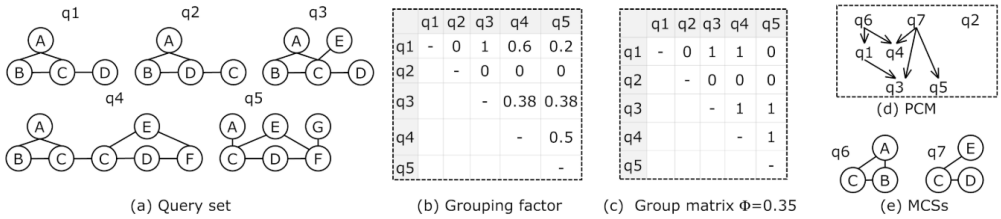


Figure 1: Building Pattern Containment Map

Figure 1. Super dependency tree set

Figure 1. TLS (T^S_3, T^S_4) ={(A-B-C), (A-C-B), (B-A-C), (D- C-E)}. We use L1(T^S_3, TLS(T^S_i, T^S_j)) to denote the number of instances in the largest instance subTree of T^S_3 corresponding to the TLSs in TLS(T^S_i, T^S_j). For example, for T^S_3, and T^S_4 in Figure 1, L1(T^S_3, TLS(T^S_3, T^S_4)) =4 and L1(T^S_4, TLS(T^S_3, T^S_4))=3。

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

Definition 2 (GROUPING FACTOR). *The grouping factor between two query graphs q_i and q_j, denoted $\mathcal{GF}(q_i, q_j)$, is defined as*

$$\mathcal{GF}(q_i, q_j) = \frac{min(\mathcal{LI}(q_i, \texttt{TLS}(q_i, q_j)), \mathcal{LI}(q_j, \texttt{TLS}(q_i, q_j))}{min(\texttt{TLS}(q_i).size, \texttt{TLS}(q_j).size)} \quad (1)$$

The grouping factor has the following properties:

(1) $0 \leq \mathcal{GF}(q_i, q_j) = \mathcal{GF}(q_j, q_i) \leq 1$.

(2) If $q_i \preceq q_j$, $\mathcal{GF}(q_i, q_j) = 1$.

(3) If q_i and q_j do not have an MCS, $\mathcal{GF}(q_i, q_j) = 0$.

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 T^S 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(T^S)，通过这个函数我们可以估计一个由句法依存树查询 T^S 生成的 SPARQL 查询 Q 的代价。当然最好有两个代价函数:

$$\text{cost}(TS) \qquad (1)$$

$$\text{Cost}(TSi, TSj) \qquad (2)$$

其中公式 1 可以给出 T^S 生成的 SPARQL 查询 Q 的代价，公式 2 可以给出由TSi, TSj 生成的 Qi 和 Qj 公共查询 Q^c 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:

$$\text{Cost}(Qi + Qj) = \text{Cost}(Qi) + \text{Cost}(Qj) - \text{Cost}(Qi, Qj) \qquad (3)$$

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

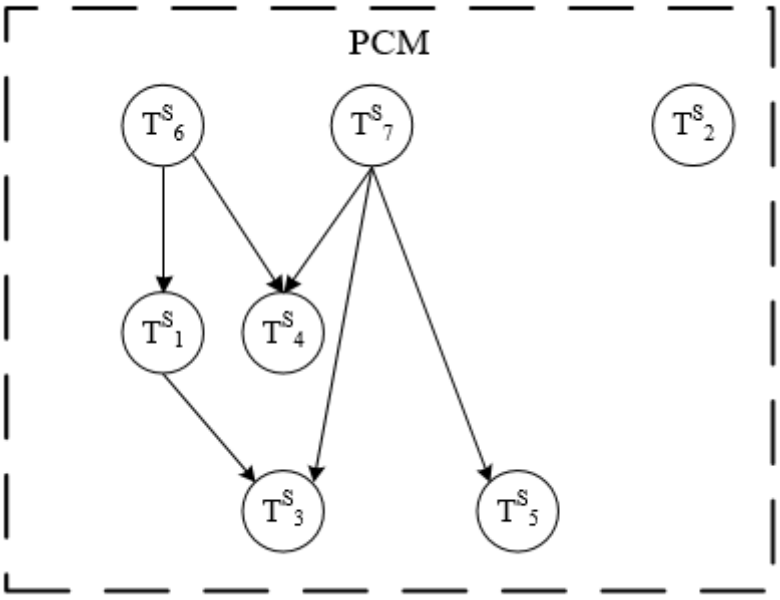


Figure 2 PCM

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 T^S_i 的边数量。

出度：PCM 中某一结点 T^S_i 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 T^S_2 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 T^S_5 就一定是和指向它的结点 T^S_4 绑定

在一组。

而针对度大于等于 2 的结点 T^{S}_{i} 就比较复杂。T^{S}_{i} 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。

3) 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）

4) 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）

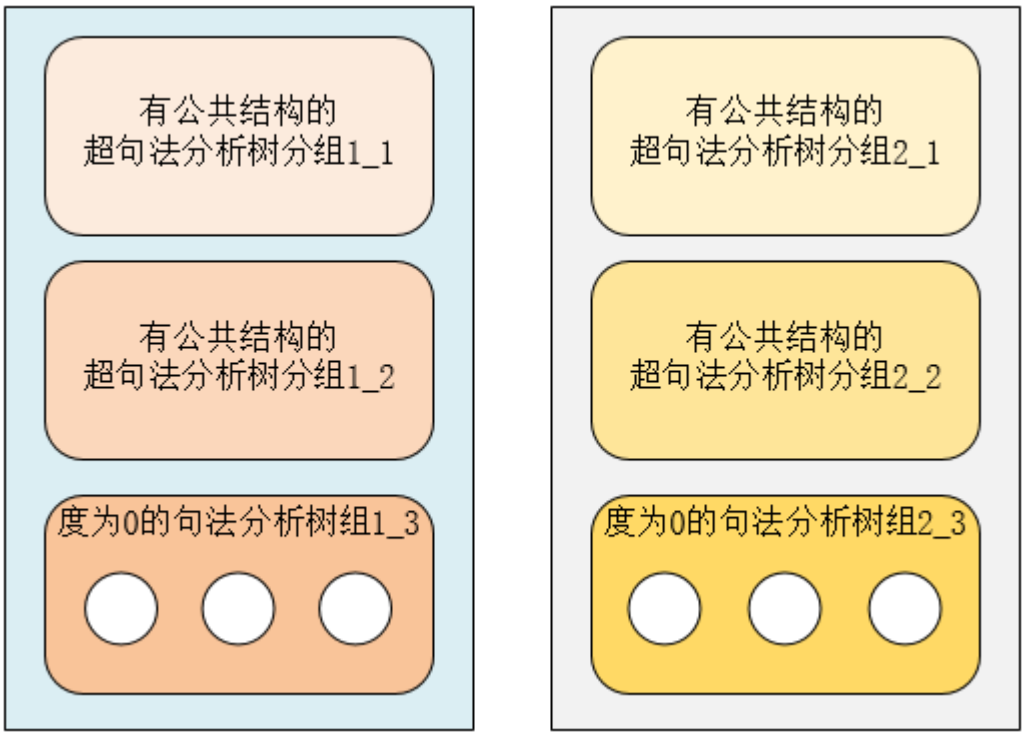
PCM 中度为 1 的结点数量小于 N/2 时

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。