# 4 算法实现和改进

本章将按照上一章提出的概念生成模型，进行算法实现，提出实际工程中将遇到的问题以及排序剪枝的解决办法，并在频繁子图的挖掘过程中图扩展的部分用前剪枝策略改进。在算法改进的基础上，在部分严重耗时的部分使用分布式计算框架进行改写，从而进一步把性能提高。

## 3.1 算法的简单实现

算法：rdf\_fpm。基于gSpan模式增长思想的RDF的频繁子图挖掘算法。

输入：DFS编码t；

RDF标注原始图集D；

最小支持度阈值MIN\_SUPPORT；

最小可信度阈值MIN\_CONFIDENCE。

输出：DFS编码树T

方法：；

对每一个可能的t调用rdf\_fpm(t,D,MIN\_SUPPORT,MIN\_CONFIDENCE,T)；

**procedure** rdf\_fpm(t,D,MIN\_SUPPORT,MIN\_CONFIDENCE,T)

(1)**if** tmin\_dfs(t) **then**

(2) **return**；

(3)将t插入T；

(4)**if** t is concept **then**

(5) **return**；

(6)DFS编码集合；

(7)扫描t对应的D一次，找出所有使t最右扩展的边e，并计算出边e的DFS编码c，与本次循环中的DFS编码s结合成新的DFS编码，将其插入C并且计算它的频度和可信度以及记录c对应的D子集D’；

(8)**foreach** C中频繁或可信的c **do**

(9) rdf\_fpm(c,D’,MIN\_SUPPORT,MIN\_CONFIDENCE,T)

(10)**return**；

其中，算法中的（1）和（2）是为了检查是否重复搜索同一个图，也即减少复制图的产生主要由这两步来完成，min\_dfs(t)的过程主要是检查DFS编码t是否满足DFS词典序，具体而言就是同时满足边序以及序列序，如果满足最小DFS编码要求，说明当前图在之前未被搜索到，并将其加入到DFS编码树T中的一个分支中；如果不满足，则停止继续搜索，直接返回进行下一个候选边的判断。算法中的（7）（8）（9）（10）是将每一个候选边都进行第一轮的频度和可信度判断，满足其中任意一个则进行下一轮的挖掘。比较特殊的一点是算法中的（4）（5），如果遇到当前边是概念边的话，则无需继续搜索和扩展，此时应该直接返回，因为已经到达树的叶节点。由此，rdf\_fpm算法成功地完成了对RDF数据图集的频繁子图挖掘，并且得到一棵DFS编码树。

进一步地，将DFS编码树转换为概念树，然后对概念树分析挖掘，即可得到最后的从RDF标注到概念集合的映射结果。但是在整个流程的实现上，会遇到一个问题是在寻找第一条边时会耗费大量的时间，因为第一条边的候选边将近有几千万条，如果使用单纯的遍历方式的话，命中率将变得极低，大部分边是没有继续扩展搜索的可能性，但是却不可避免地要处理一遍。也正因此，对第一条边的候选集合选取需要做一次预处理优化，也就是下一节提到的排序剪枝优化。

## 3.2 排序剪枝优化

排序剪枝优化指的是将有可能成为第一条边的候选集合先筛选出来，然后对这一集合进行直接搜索扩展，而忽略其它存在的边集合。但是，如何将这一可能的候选边筛选出来呢？

实际上，我们可以做一次排序。将所有顶点属性，边属性出现的频度记录下来，得到一组顶点属性和频度的数据和一组边属性和频度的数据。然后根据频度分别对两组数据从大到小排序，这时每组数据又多了一个值，频度在序的位置rank，因此我们将属性用rank替换，对图集D进行一次重新标注，现在每条边每个顶点的属性不再是本身的属性而是在整个属性出现频度序的位置。

由此，有了排序rank的这个新添信息，我们可以着手于做剪枝优化的工作。根据Apriori性质：频繁项集的所有非空子集也必须是频繁的。反过来说，一个不频繁的项集它的所有超集也是不频繁的，这一性质也满足一类特殊的性质，反单调性。因此，对于不频繁的顶点或是不频繁的边，我们可以直接采取剪枝策略，从而减小扩展时被搜索到但频度又不满足的可能性，大大地提高命中率。

补充图，说明剪枝和二次rank

但是这仅仅提高了命中率，第一条边的选取的时间复杂度仍然是，这是因为选取一条边需要三个变量，变量m控制顶点的rank取值，变量n控制边的rank取值，其中m从0到顶点频度满足MIN\_SUPPORT的最大rank变化，n从0到边频度满足MIN\_SUPPORT的最大rank变化。我们可以通过进一步优化将选取时间复杂度降为线性时间，只控制边这一个变量，在记录频度的时候将每条边的信息包括起始顶点，终结顶点，边属性都记录下来，这样对这类边也做一次排序，满足频度大于MIN\_SUPPORT的边被选作第一条边的候选边。事实上，这样做命中率实际为100%，因为满足频度要求的每条边都至少是DFS编码树的一个分支。

## 3.3 前剪枝策略

除了在第一条边选取候选边优化之外，还可以在检查重复这一环节进行优化。本章第一节介绍了算法的简单实现，其中算法中的（1）（2）是为了防止复制图即重复图的产生，通过检查该DFS编码是否是当前图的最小DFS编码，即可知道该图是否满足DFS词典序，进而发现该图是否是复制图。但实际上，我们还有一种方法可以做到这一点，而且无需再检测这个DFS编码是否最小的过程，这个过程可以被简化。

这个优化策略我们称为前剪枝策略，这里的剪枝其实和算法中（1）（2）的剪枝是一个分支，但是这个步骤提前到扩展候选边的环节，所以我们称之为前剪枝。在扩展候选边的过程中，我们发现只要控制一个图中边的发现顺序，就可以防止同一复制图被搜索到两次，因为复制图的产生最大的原因就是候选边是个集合，缺少顺序，因此搜索到的图可能是同一个图但是边的顺序颠倒了，因此通过固定边的发现顺序，颠倒的情况便不复存在。

这个顺序我们可以复用DFS词典序，或者自己定义一个顺序也完全可以。

补充图说明

## 3.4 分布式改写

## 3.5 本章小结