

# 函数依赖挖掘实验报告

---

- 张迁瑞 2015013226
- 高敬涵 2015013194

## 实验环境

---

(是否要统一实验环境？)

- TANE算法: MACBook Air, CPU 1.6GHz Intel Core i5, 8GB内存。
- DFD算法: Windows, CPU 2.5GHz Intel Core i7, 8GB内存。

## 编程语言

---

- TANE: C++, 使用Xcode开发。
- DFD: C++, 使用Visual Studio 2017开发。

## 实验过程

---

### TANE实现

实现TANE是一个不断优化的过程，首先按照论文中对于TANE的描述实现了算法，发现运行结果不对，原因是论文中对于Prune的描述不准确，删掉后一个剪枝方法后得到了正确结果。在data.txt上的运行时间为150秒左右。

之后通过分析发现，对运行时间影响最大的是getPIPProduct函数，于是修改了getPIPProduct中的数据结构，使用vector代替map，运行时间下降到45秒。

后来，我们将getPIPProduct中的set更改为vector，运行时间进一步下降为10秒。之后，我们又尝试使用数字的二进制表示代替set表示属性集合，将运行时间降低为7.5秒。

### DFD实现

实现DFD是一个纠结的过程，因为论文中的描述并不准确，而且有些地方还有错误，所以在实现的过程中要反复研读DFD的本质机理，在充分了解算法过程的前提下才能完整的将DFD实现出来。

最后初次正确的结果跑了17秒左右，对其中的关键代码进行了优化，将上一层的partition充分利用，尽量减少compute partition的次数，最后最快跑出了10.5秒的速度。

# 实验结果

---

TANE：运行时间7.5秒 DFD：运行时间10.5秒

## 实验结果分析

---

### TANE结果分析

从结果来看，对整个程序运行时间影响最大的是getPIProduct函数执行的次数和getPIProduct一次执行所用的时间。

getPIProduct每次执行要遍历其子集中的全部集合，再在集合的基础上遍历集合中的元素，最坏情况下是 $O(N)$ 的，运行速度取决于stripped\_partition中元素的个数。所以使用stripped\_partition(删除所有长度为1的集合)可以显著减少运行时间。

每次有一个新的candidate被加入到level集合中时，getPIProduct都要被执行一次，所以剪枝的作用也非常大。

从数据对实验结果的影响来看，数据行数的影响更大。在本次实验中，TANE一共只执行了六层，后面更加复杂的属性组合都没有遍历到。

### DFD结果分析

与TANE的分析相似，在程序运行的时间上，影响最大的是计算partition的函数的执行次数，如何尽量减少此函数的执行是程序优化的关键，而DFD由于它本质机制的影响，其对上一层的已经计算好的partition利用程度没有纯level-wise的TANE高，所以在小数据上的表现并不如TANE。

但是我认为DFD的最大优势是在应对函数依赖数量特别多的时候，其由于能够最大程度的减少无用的candidate，从而从另一个角度减少partition的计算次数。

在属性集合的表示上，我们利用int的比特位来表示属性是否出现在属性集合中，在计算超集和子集的时候将最大程度的提升运算速度，但是从长远的角度考虑，使用Bitmap是更好的选择。

注：最后的测试请采用TANE作为我们的最终算法。