数据库函数依赖挖掘作业

2015013212 2015013222

候建国 陈超

1. **实验内容：**

实现课上学习的一种函数依赖挖掘算法。

1. **数据集：**

数据集包括15个属性名，共10万条以上数据。

**三. 实验环境:**

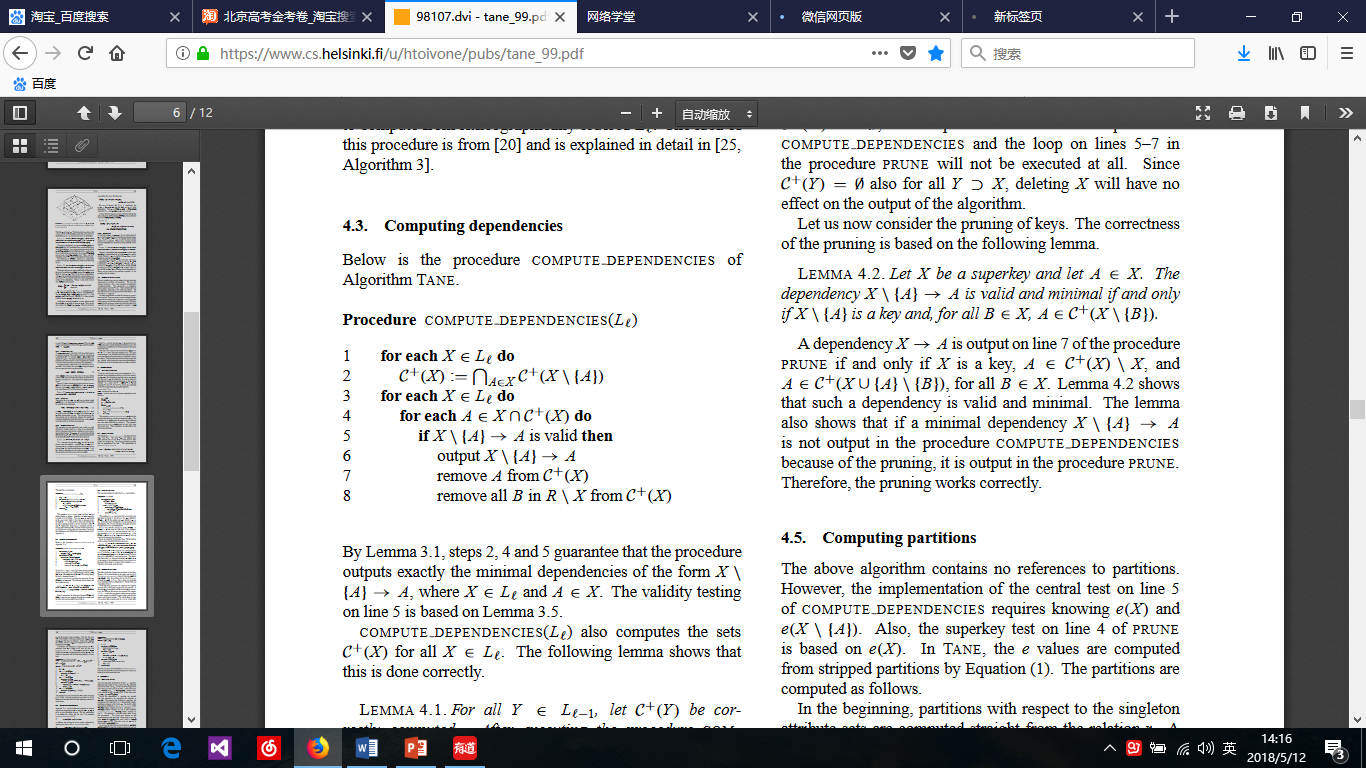
开发测试环境：

* 操作系统：Windows 10
* IDE: Visual Studio 2012/2015
* 编程语言：C++

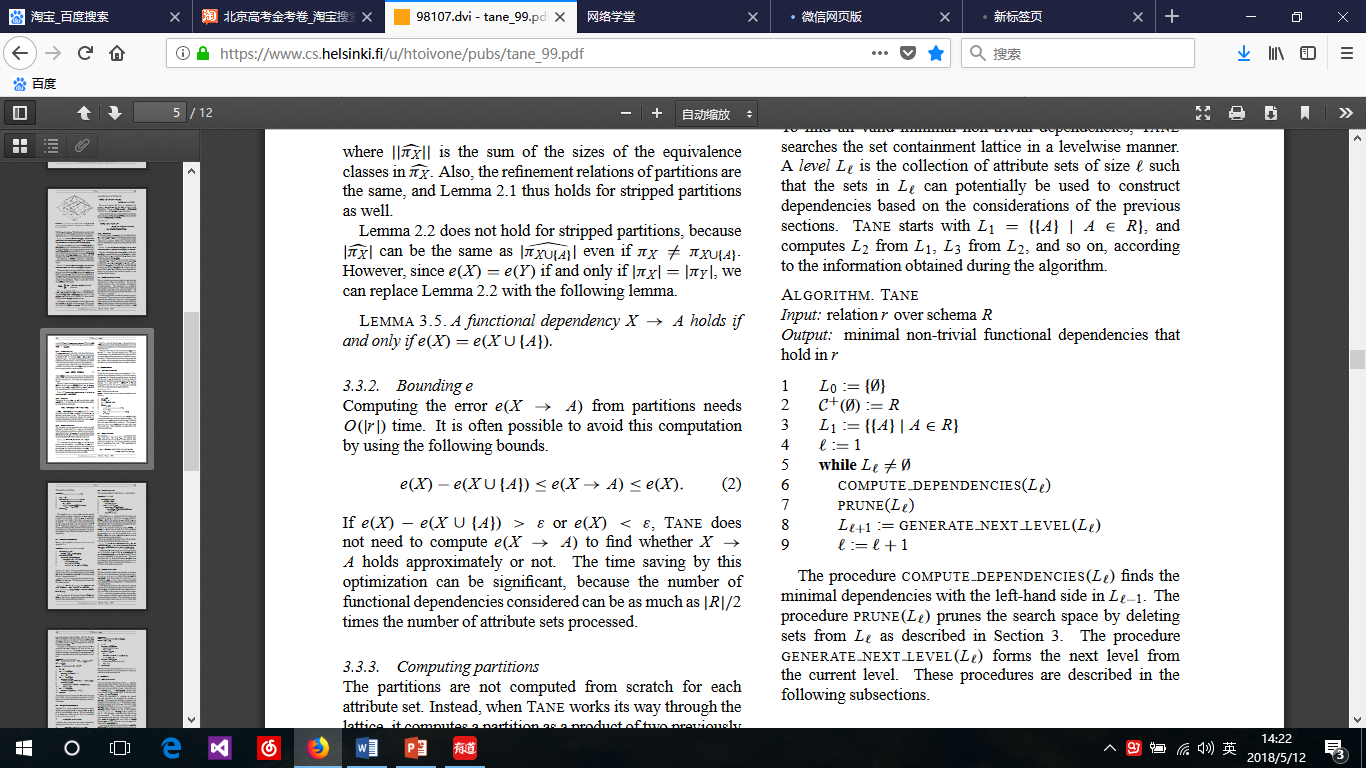
**四. 实验过程**

1. 算法的选定：本次实验我们综合了算法的难易程度、效率高低、以及小组同学对算法的了解最终选定了TANE算法，该算法不仅仅在效率上比较快，同时也适合更大的数据集。 但之后我们接着尝试了DFD算法。但是由于时间问题没来及的完成。因此这里仅说明TANE算法。

2. 算法的理解：TANE算法的目的是找到所有有效的最小非平凡的函数依赖。它对数据几何组成的格子结构进行分层，然后对每一层计算函数依赖，计算完之后对该层进行剪枝，接着更新到下一层。对于计算函数依赖的具体步骤，显示如下:



该函数通过上一层的找到最小函数依赖。这里2到5行保证了程序的输出一定是最小的函数依赖。其中第5行的完整性检验利用了如下定理：



3. 算法的思想：

（1）数据集存储：使用一个vector<vector<int>>（FT\_table）的table存储数据集。

（2） 属性的存储：我们使用二进制的方式来存储属性，比如1存储属性1,10存储属性2,100存储属性3，以此类推；而属性组合就可以通过二进制计算来得到，比如25组合属性可以通过10+10000=10010来表示，这样做可以不必使用数组来存储每个属性而减少空间浪费，同时还有利于组合属性的计算。

（3）每层的属性存储：用set<int>来存储一层的属性，vector<set<int>>（FT\_levels）存储格子结构的每层属性。

（4）RHS+存储：使用一个的数组vector<int>（FT\_RHS）来存储。

（5）π存储：对于每个属性，使用vector<vector<int>>来存储该属性不同分区的实例行，vector<vector<vector<int>>>（FT\_partitions）将所有属性包含在内；使用vector<int>（FT\_exists）来保存每个属性的分区信息是否已计算过，计算过的便不用再次计算；使用vector<int>（FT\_piLen）来存储每个属性的分区行的总长度。

4. 算法的主要实现：

在算法的实现过程中，我们参考了相关论文

TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies

中的算法的实现步骤。

我们构建了一个FDTANE类，用于数据导入导出以及算法的具体实现。

初始情况下，我们调用该类的构造函数FDTANE(in, out, n)读入数据，构建FT\_table，然后初始化每个存储结构，并初始化格子层状结构的第一层的FT\_levels和FT\_RHS。

本程序中的主要函数包括：

generateFD()：逐层计算，得到最小函数依赖并输出，同时计时以观察效率。

generateNextLevel(int level)：计算level层的属性集。

computeDependencies(int level, ostream& outputstream)：计算level层的函数依赖项并输出。

**五. 实验结果分析**

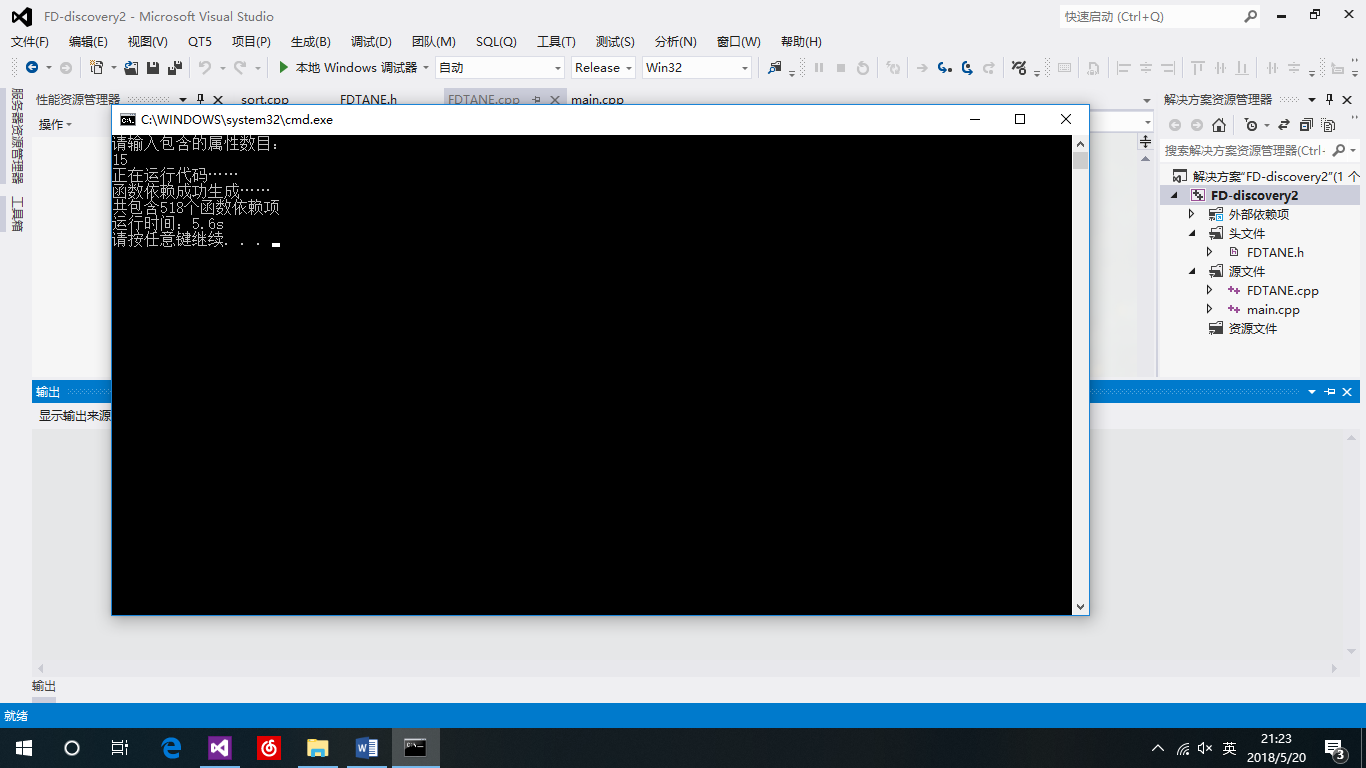
我们使用测试数据test\_data.txt与data.txt进行结果分析

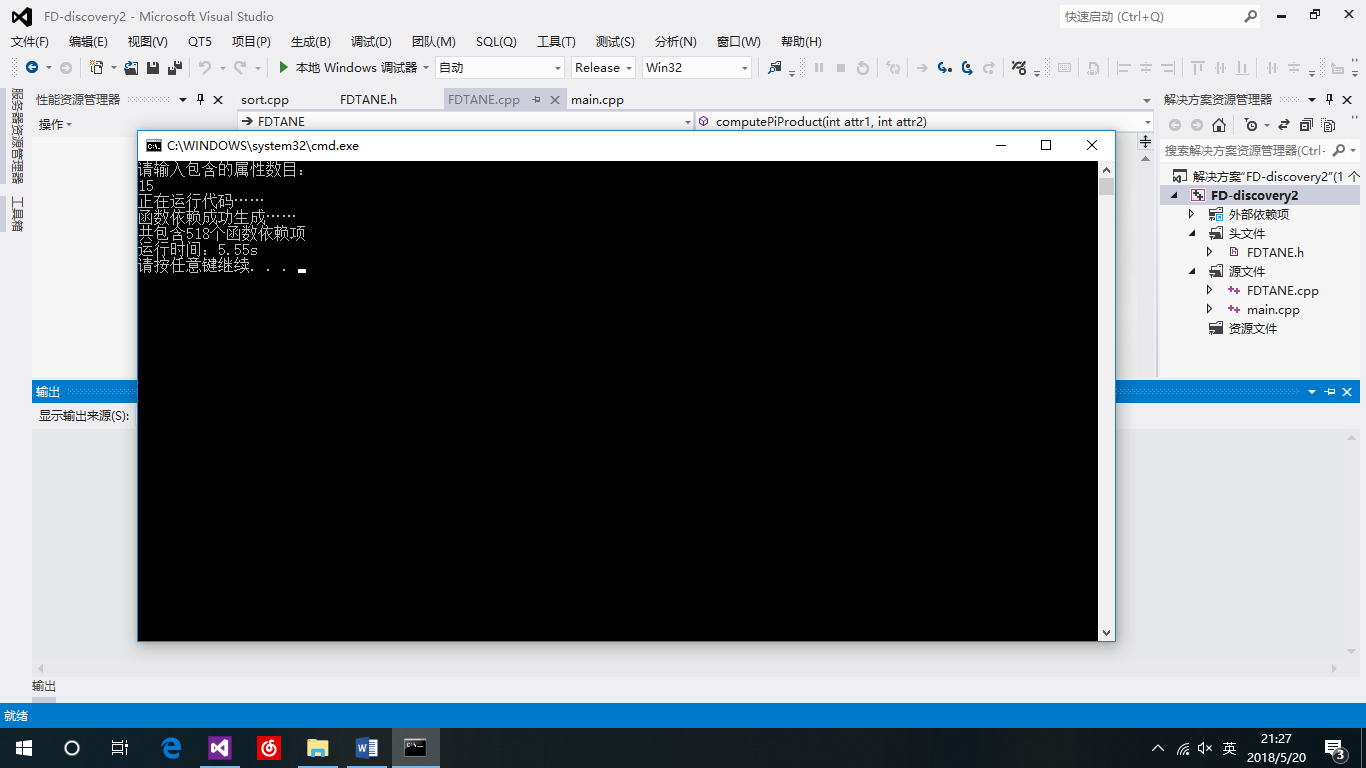
1. test\_data.txt

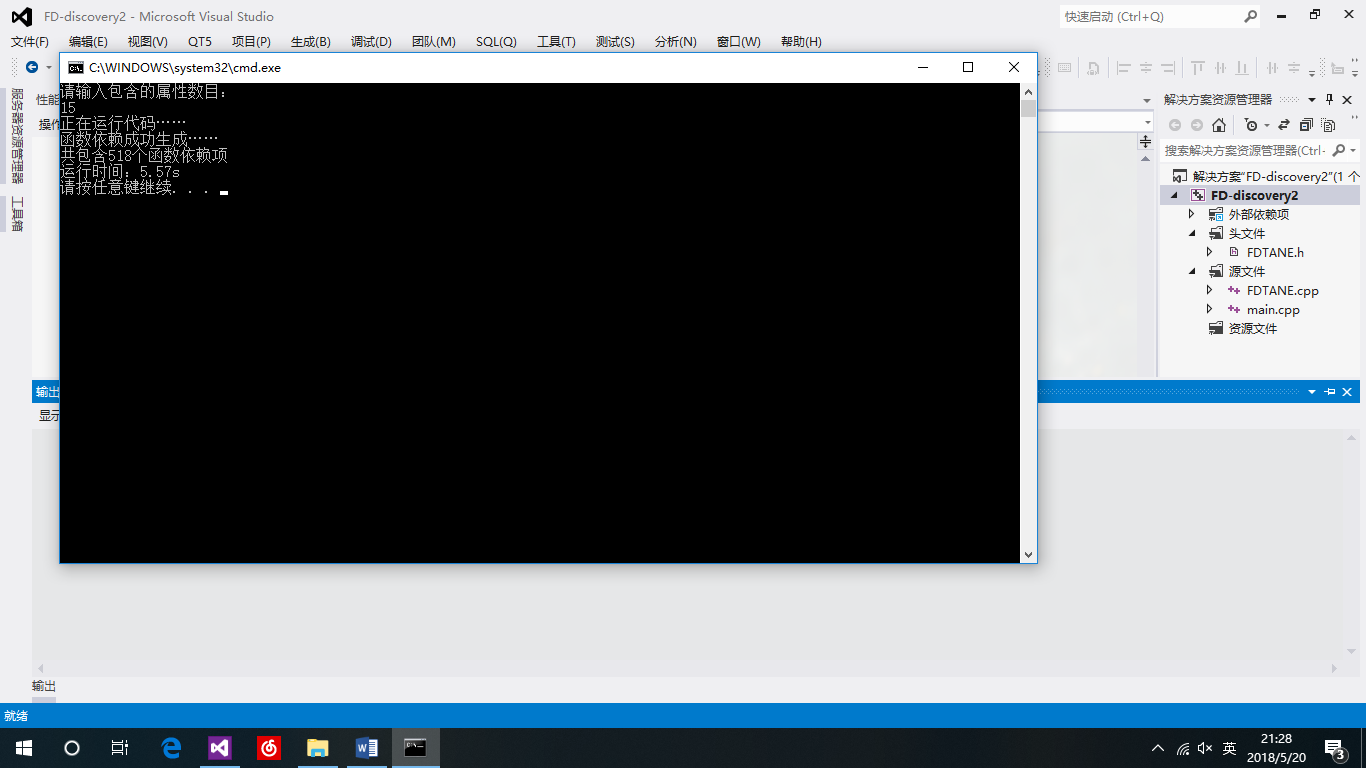
对于测试数据集，我们主要侧重于结果的正确性，经过与助教提供的结果比对，我们的程序结果是正确的。对于该数据集，总共产生了109个函数依赖项，平均用时为0s.

1. data\_txt

对于大数据集，我们主要侧重于得出结果的效率。该数据集总共产生了518个函数依赖项。我们总共进行了3次测试，最终取平均得到平均花费时间为5.57s，以下为截图：







因此，对于10万+的数据量的处理，该算法还是呈现出较好的计算效率。

PS:

在之后的工作当中，我们发现DFD算法是一个比较新兴的算法，而且通过论文的阅读，可以看出DFD是要比传统的TANE算法效率高。因此我们尝试再实现一个DFD算法。

通过论文的阅读，大致得到了如下思路：

先将属性集合中的主键剔除，接着对剩下的属性集合(RHS)遍历，得到LHSs.从而得到最终结果。我们实现的算法的主要函数包括：

void DFD::generateFD() 计算函数依赖的整体函数

void DFD::findLHS(int attr) 寻找当前属性attr的所有LHSs，先生成种子，对种子中的某一个node判断或更新其类型,并得到下一个node继续执行

int DFD::pickNextNode(int attr, int node) 根据当前node寻找下一个node

vector<int> DFD::generateNextSeeds() 通过mindeps和maxnondeps生成下一批种子，继续在函数findLHS中执行

我们的代码尚不能运行，因为时间问题其中的bug来不及调试。但函数的基本结构框架和具体功能已实现，因此我们将DFD的代码一并提交，并在之后的时间里，我们会继续将其完善。