2015-11

现代数据库系统

——GSkyline算法实现与评估

杭天梦 2015213527 hangtianmenglisa@163.com

王需 2015311975 wangxu.93@icloud.com

康荣 2015311965 kr11@mails.tsinghua.edu.cn

**GSkyline算法实现与评估**

在本次作业中，我们小组实现了***Finding Pareto Optimal Groups: Group-based Skyline***中提出的PointWise Algorithm 和 UnitWisePlus Algorithm，并对算法实现的正确性和效率进行了评估，通过比较两种算法在不同数据集的表现，初步提出了进一步改进的思路，具体实现将在第二次作业中完成。

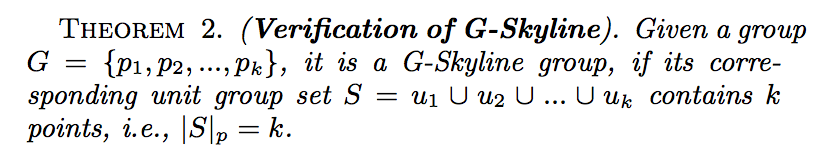
# 背景

Skyline的关键是它包含了所能支配其他点的“最好”的点。但是有时候我们想要搜索一组备选数据而不是唯一一个数据，这时就需要引入GSkyline概念。给定组的长度k，它能选出点集中最优的组，且该组不会被其他组支配。

## PointWise算法

该算法的主要思想是每次在每一层动态生成候选的排列树，同时也要将非G-Skyline的候选树尽可能多的删除。对于每一个节点，我们会为它存储一个尾集(tail set)，尾集用来存储比此节点序号大的所有节点。这样，每次从节点的尾集中挑选一个合适的节点加入到当前的集合中，便可以生成新的集合。在预处理后，根节点包含的集合是一个空集，并且它的尾集是前k层所有节点的集合。

## UnitWisePlus算法



根据定理2，我们可以知道G-Skyline组可以由单元组（unit group）的集合组成。

前面的PWise算法是一次添加一个点，而UWise+算法一次添加一个组，接下来的思想和PWise基本相似。值得一提是，UWise+是UWise算法的加强版，它对所有的单元集做了一个逆序，因为节点的序号越大，就意味着其所在的层数越大，那么它就会有很多父亲节点，因此，它的单元组的元素个数就更大。这样的好处是：不符合要求的单元组会在算法执行前期被删掉，因此增加了算法的执行效率。

# 实验环境

|  |  |
| --- | --- |
| 操作系统 | Windows8.1 |
| 处理器 | Intel(R) Core(TM) i7-4790K CPU 4.00GHz |
| 内存 | 8．00GB |
| 编译环境 | Visual Studio 2013 |

# 使用指南

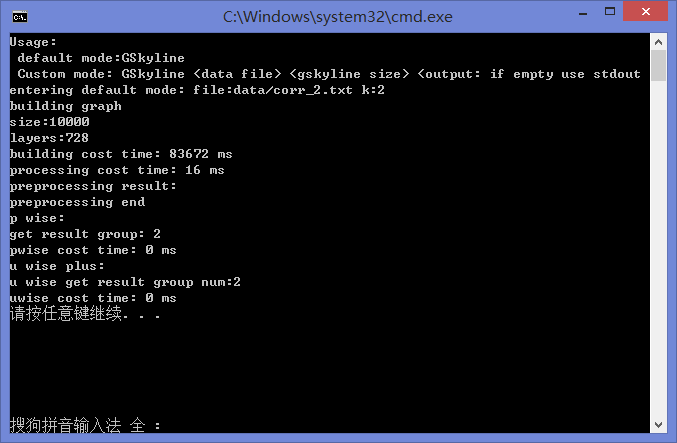
算法使用方法介绍：

1. 在控制台中，输入

“GSkyline.exe 数据文件路径+文件名 k的值”

如“GSkyline.exe E:/input.txt 2”,结果会输出到控制台。

1. 输出值解释



Size:数据量大小

Layer：生成的层数

Building cost time:建立有向图所花的时间

Processing cost time:预处理所花的时间

Get result group: pwise算法所得的最优组的组数

Pwise cost time：pwise所花费的总时间

U wise get result group num: Uwise+算法所得最优组的组数

Uwise cost time： Uwise+算法所花的时间

# 改进详述

PWise方案改进

UWise+方案改进

# 性能比较

如下表所示，PWise+算法为PWise的改进算法，UWise++为Uwise+算法的改进算法。

表1-算法运行表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 文件 | k | Pwise+运行时间(ms) | Uwise+运行时间(ms) | Uwise++运行时间(ms) | 建图时间(ms) | 预处理时间(ms) | 组数 |
| anti\_2.txt | 1 | 0 | 0 | 0 | 484 | 0 | 139 |
| anti\_2.txt | 2 | 0 | 0 | 0 | 484 | 0 | 9717 |
| anti\_4.txt | 1 | 0 | 0 | 0 | 421 | 15 | 8960 |
| anti\_4.txt | 2 | 9765 | 12870 | 63 | 422 | 0 | 40137097 |
| anti\_6.txt | 1 | 0 | 0 | 0 | 421 | 16 | 9386 |
| anti\_6.txt | 2 | 10686 | 14196 | 63 | 421 | 0 | 44044277 |
| anti\_8.txt | 1 | 0 | 0 | 0 | 734 | 15 | 5378 |
| anti\_8.txt | 2 | 3432 | 4477 | 31 | 764 | 0 | 14460071 |
| corr\_2.txt | 1 | 0 | 0 | 0 | 1373 | 0 | 1 |
| corr\_2.txt | 2 | 0 | 0 | 0 | 1139 | 0 | 2 |
| corr\_4.txt | 1 | 0 | 0 | 0 | 905 | 0 | 35 |
| corr\_4.txt | 2 | 0 | 0 | 0 | 983 | 0 | 625 |
| corr\_6.txt | 1 | 0 | 0 | 0 | 811 | 0 | 404 |
| corr\_6.txt | 2 | 16 | 31 | 0 | 796 | 0 | 81639 |
| corr\_8.txt | 1 | 0 | 0 | 0 | 749 | 0 | 1520 |
| corr\_8.txt | 2 | 265 | 359 | 0 | 749 | 0 | 1155102 |
| inde\_2.txt | 1 | 0 | 0 | 0 | 842 | 0 | 10 |
| inde\_2.txt | 2 | 0 | 0 | 0 | 796 | 0 | 47 |
| inde\_4.txt | 1 | 0 | 0 | 0 | 811 | 0 | 172 |
| inde\_4.txt | 2 | 0 | 0 | 0 | 780 | 0 | 14814 |
| inde\_6.txt | 1 | 0 | 0 | 0 | 687 | 0 | 887 |
| inde\_6.txt | 2 | 94 | 109 | 16 | 702 | 0 | 393417 |
| inde\_8.txt | 1 | 0 | 0 | 0 | 733 | 0 | 2794 |
| inde\_8.txt | 2 | 905 | 1170 | 0 | 687 | 0 | 3902795 |

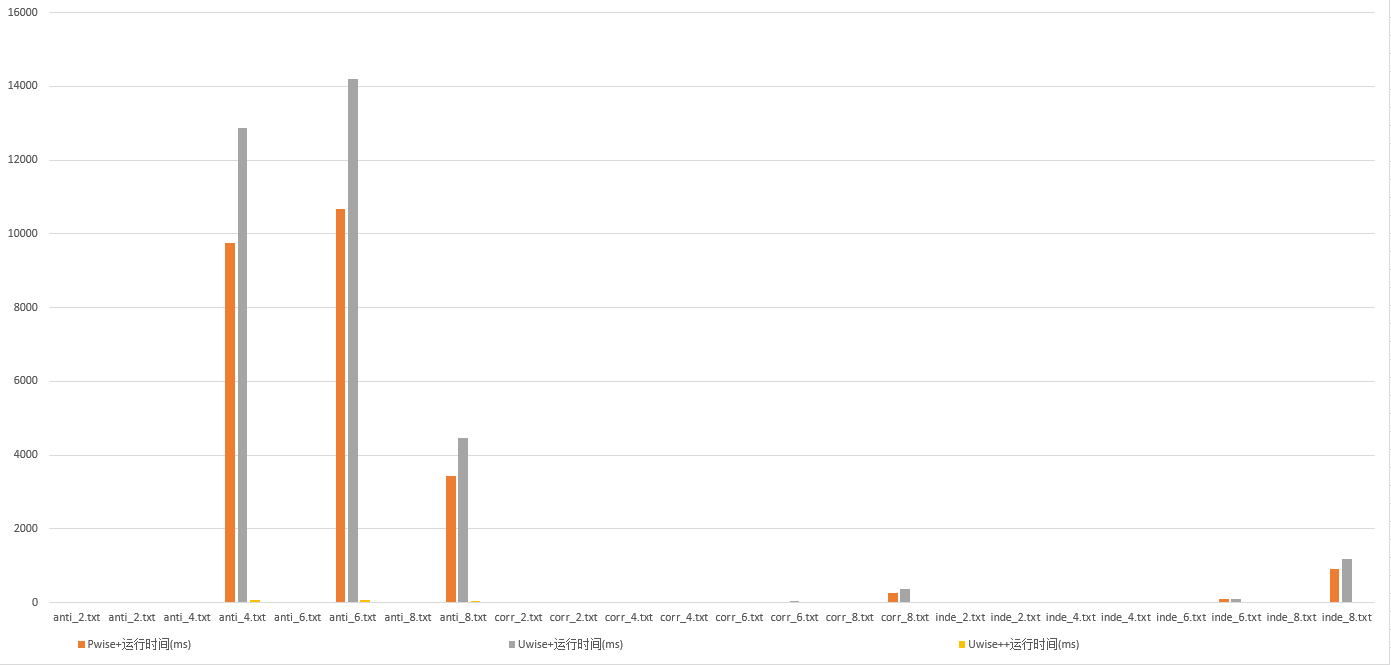


图1 不同数据集三种算法的运行时间

如图1所示，x轴为文件名，y轴为三种算法运行所花的时间。由图可得，UWise++花费的时间均为最小，Pwise+所花的时间小于UWise+时间，PWise算法的优化方案也有了明显的效果。

从图1也可得，对于不同的数据集，算法花费的平均时间也不同。当数据维度相同的时候，anti数据集花费的时间普遍高于corr和inde数据集。由图可知，当数据集的维度等于8时，所花费的时间为anti>index>corr。尽快如此，Uwise++算法也体现出很出色的效果。

当k=1时，不同数据集上所有的算法执行所花时间均小于1ms,当k=2时，不同数据集上各个算法的效率开始拉开了差距。当k≥3时，所花的时间较多，但UWise++算法依然具有优势，如，当数据集为anti\_4，k=4时，T=188371ms。

# 结论

上一阶段，我们实现了Pwise算法的改进与Uwise+算法。Pwise+算法（改进后的Pwise算法）的性能明显优于朴素Uwise+算法。

这一阶段，我们实现了Uwise+算法的改进。Uwise++算法（改进后的Uwise+算法）在运行效率上有了显著的提升，并优于Pwise+算法。

整个过程大家均付出了努力，在讨论过程中碰撞出了思想的火花。大家也提出了很多新的思路与想法。

感谢助教批阅！