**Group skyline 实验报告**

丁霄汉 2017312365 吴超月 2017213865 罗  瑶  2017213866

# 实验目的

实现论文《Finding Pareto Optimal Groups: Group-based Skyline》中的G-Skyline算法。

# 实验任务

主要包括实现文中生成skyline layers的算法并对结果集进行预处理生成directed skyline graph、point-wise算法的实现、unit group-wise+算法的实现。

注：由于unit group-wise+在unit group-wise算法基础之上进行了进一步的剪枝，因此在实验中我们直接实现了unit group-wise+算法。

## 2.1生成DSG

生成skyline layers并构造后续算法所需的数据结构directed skyline graph。

### 2.1.1基本思路

主要分为以下几个步骤：

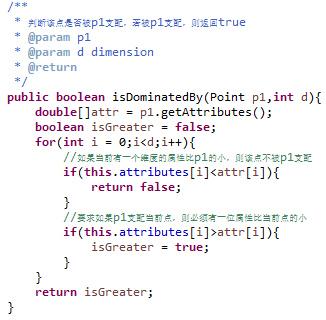
1. 获得给定的数据集后，按照第一维度属性值升序排序；
2. 判断属性维度，如果等于2则直接使用二维下的方法进行skyline layers生成，否则进入高维属性下的skyline layers生成方法；
3. 获得相应的k条skyline后，对每个节点利用论文中的预处理思想去掉不可能出现在group skyline中的点，生成预处理后的DSG。

### 2.1.2代码实现

1. **Point类**

Point类用于存放从文件中读入的每条数据，成员变量包括用于标记当前数据的标号以及存放各个维度数据的double类型数组。Point类实现了Comparable的compareTo方法，使得Point可以按照其第一维度属性值进行升序排序。

此外，Point类包含一个isDominatedBy方法利用定义判断当前点是否被给定的点所支配：如果当前点某个维度上的属性值比给定的点小，则不被其支配；如果所有维度上的属性值都不比给定的点相应的值小，且至少存在一个维度上的属性值比给定点的大，则该点被给定的点所支配：



1. **DSGNode类**

DSGNode类用于表示DSG中的每一个节点，成员变量包括用于标记当前数据标号的pointIndex、记录数据所在的skyline层号的layerIndex、存放各个维度数据的double类型数组attributes、记录其所有父节点的列表parents、记录其所有子节点的列表children以及记录属性维度个数的整形变量d。

1. **ProcessResult类**

ProcessResult类保存程序执行过程中需要记录的数据，包括一个存放所有供后续算法使用的DSGNode列表、存放在预处理中已经找到的符合要求的group以及存放最终结果集的列表。

1. **DSGGenerator类**

DSGGenerator类为生成directed skyline graph的主要处理类，成员变量包括所生成的group skyline中的成员个数k、传入的待分组数据集、记录生成的k条skyline的列表以及每条数据的维度个数d。

在调用generateDSG方法生成DSG时，该方法首先调用成员方法generateSkylines生成k条skyline。在generateSkylines方法中，首先对数据集按照第一维度属性升序排序，然后判断属性的总维度，若为2，则调用generate2D方法利用论文中的algorithm1生成k条skyline，否则调用generateHighDimension方法使用简单的遍历生成k条skyline。在获得k条skyline后，利用论文中的方法对数据集进行预处理，并最终生成供后续使用的DSG。

在调用generateSkylines方法生成k条skyline后，对每个节点进行预处理，筛选掉不可能出现在group skyline中的点，即判断其unit group的大小是否大于k，如果大于则抛弃这些节点；再对每一个点遍历在其所在skyline之前的所有skyline上的点，判断这些点是否支配该点，若支配，则将该点加入对应点的children列表中，相应的将这些点加入该点的parents列表中，最终生成DSG。

generate2D方法即对论文中algorithm1的实现，在其中调用了binarySearchLayer方法加快对给定点所属skyline层号的判断。此外，在生成新的skyline时判断当前skyline的层数，若大于k，则不再继续生成。

generateHighDimension方法即将论文中的algorithm1算法应用到高维属性中，此时无法利用二维情况下的单调性，即需要遍历每个已加入skyline的节点进行判断新加入的节点属于哪个skyline。

## 2.2 point-wise算法实现

从预处理后得到的DSG出发，DSG中的DSGNode已经按照index由小到大排列，用point-wise方法计算group skyline

### 2.2.1基本思路

point-wise方法的基本思路就是从空集开始，逐层生成大小为1，2…k-1，k的G-skyline groups。当已经得到大小为i的G-skyline groups之后，遍历每个大小为i的G-skyline group，每个G-skyline group都有一个尾集合，尾集合包含所有index大于G-skyline group中所有点的index的点，遍历尾集合中的点，向G-skyline group中增加尾集合中的点构成大小为i+1的candidate group，然后检查该candidate group是否是G-skyline group，如果是，就将candidate group加入大小为i+1的G-skyline groups。point-wise尽可能的在每一步剪枝。以下是point-wise的几个剪枝方法：

1. Subtree Pruning：一旦某个大小为i的candidate group被检查之后发现不是G-skyline group，那么往其中加入任何点都不能构成大小为i+1的G-skyline group.所以我们可以直接舍弃这个点不存储
2. Tail Set Pruning：

A、G-skyline group中如果包含某个点p，那么点p的父母节点一定也在这个G-skyline group中。所以我们可以对G-skyline group的tail set进行删减，我们首先计算得到G-skyline group中所有节点的孩子节点集合，tail set中的点如果不在孩子节点集合中或者属于第一层skyline则直接从tail set中删除该点。这样可以减少candidate groups的大小。

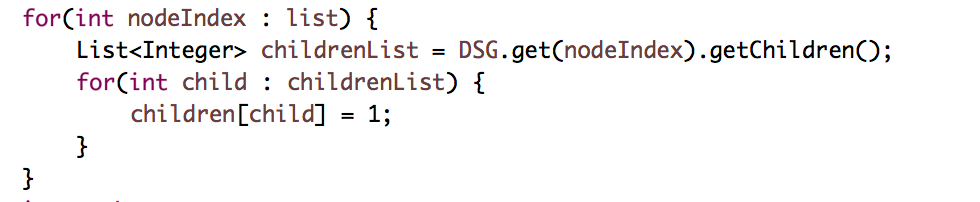
B、假设G-skyline group中节点的最大层数为i，则加入的点p必须来自前i+1层。所以tail set中的点如果层数大于i+1也被直接删除。

总的来说，point-wise包含三个循环，最外的循环每循环一次生成一层的所有G-skyline groups，层数从1到k；在生成每一层的G-skyline groups时，循环遍历所有上一层得到的的G-skyline groups，遍历每一个上一层的G-skyline group时，对tail set进行剪枝；然后遍历剪枝后的tail set，每次遍历得到一个candidate group，检查是否为G-skyline group，如果是，加入集合，否则直接舍弃。

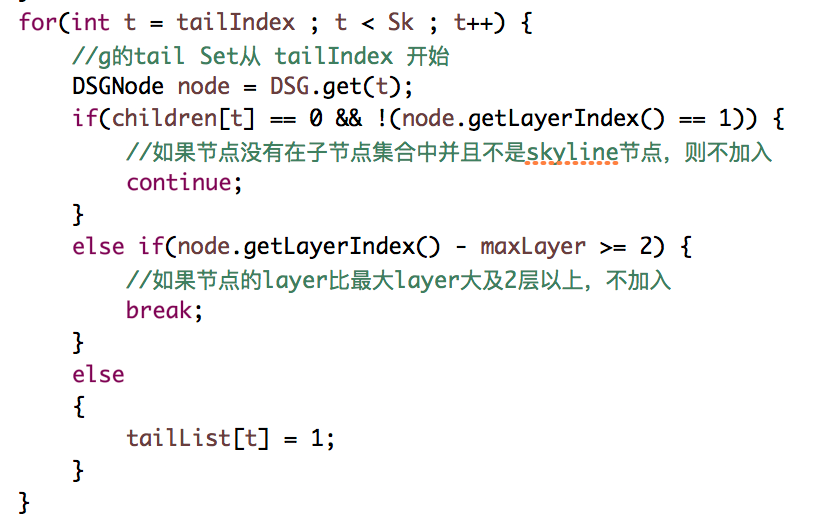
### 2.2.2代码实现

代码使用java实现。

1. 每一个G-skyline group使用List<Integer>表示，每个节点使用其index代表。最后得到的大小为k的G-skyline groups用List<List<Integer>> groupListNew表示。
2. 使用int[] children来统计每个G-skyline group的孩子集合，下标i中的整数为1代表index为i的节点是G-skyline group的孩子，为0代表不是孩子。具体代码实现如下，遍历G-skyline group的每个节点的孩子节点列表，将children数组对应于该孩子节点的下标index的整数设为1。



1. 使用int[] tailList来统计删减过后的尾集合。下标i中的整数为1代表index为i的节点是没有删减的节点。具体实现如下：



1. 遍历tailList，如果下标为i的位置整数为1，将index为i的节点加入G-skyline group，检查是否为新的G-skyline group，这里使用了一个检查是否为G-skyline group的检查方法：检查candidate group的unit group（自己的父母节点加上自身的集合）的大小是否等于candidate group自身的大小，这里使用java的数据结构Set／HashSet来去重检查。具体代码实现如下：



最后groupListNew中存储所有大小为k的G-skyline groups，groupListNew加上DSG预处理阶段得到的unit大小为k的节点集合perfectNodeList就是所有的解。

## 2.3 unit group-wise+算法实现

从DSG出发，用unit-wise方法计算group skyline。

### 2.3.1基本思路

主要分为以下几个步骤：

1. 以上一步骤输出的DSG为输入，简化数据结构，构造RefDSG，即Reference-based DSG。这是因为上一步输出的DSG的节点DSGNode是通过List中的下标来存储其父母节点和孩子节点的，而在unit-wise方法中，在涉及unit group的操作时需要频繁查询和合并许多节点的父母节点和孩子节点。将DSGNode转换为基于引用的RefNode，用HashSet保存节点的父母节点和孩子节点的引用，并抛弃无用属性，可以提高效率。另一方面，将排序后的unit group和node作为RefDSG的成员变量，可以提高抽象层次，提高可维护性。
2. 按照论文中的实现方式，将nodes和unit groups按照原本的索引反向排序。
3. 构造仅由单一unit group组成的初始groups列表，即singleUnitGroups。
4. 对每一个初始group：
5. 检查Glast。
6. 以该group为初始节点，构建枚举树，并按照论文中的方法进行剪枝。在这里我们结合java的语言特性做了一些改进。

最终输出结果。

### 2.3.2代码实现

1. **RefNode：**

基于引用的DSG Node。用originPointIdx表示原本的节点索引，parents表示其所有的祖先节点，children表示其所有的后代节点。这里沿用了论文中parent和children的称谓，虽然这个命名不是很精确。这里使用HashSet是为了提高查询和合并的效率。

1. **Unit类**

单个unit group，记录其起始节点和所有长辈节点。

1. **UnitGroup类**

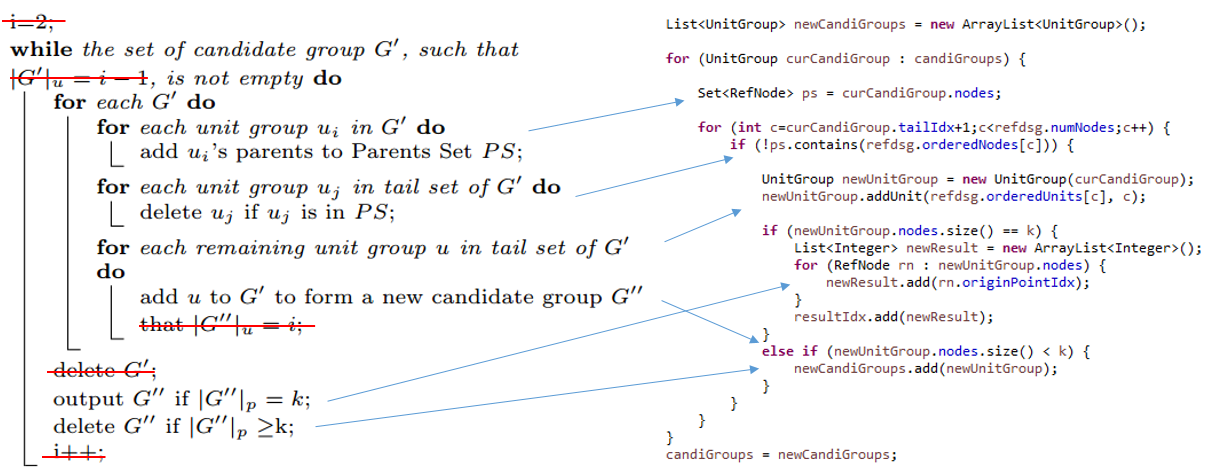
由unit group组合而成的group。记录其在枚举树tail list中的位置和包含的节点。

1. **RefDSG**

基于引用的DSG。保存排序的nodes和units。

1. **UnitWise类**

负责应用论文中描述的unit-wise方法进行计算并输出结果。

在具体的实现上，我们结合java的语言特性对论文中的方法进行了优化。以下面这段代码为例。

论文中所说的一个node的parent set其实可以等价替换为对应unit group的nodes集合。这两个集合的差集就是这个node本身，这一差别对剪枝没有影响。这个集合其实我们已经在之前求出来了，因而大大提高了效率。

在求tail set的时候，我们没有像论文中一样“删去在parent set中的unit group”，而是“只添加不在parent set中的unit group”。考虑到java中ArrayList的删除操作效率低下，LinkedList的插入操作效率又低于ArrayList，这样做可以使我们既使用ArrayList又避免删除操作，提高了效率。

文中使用变量i来索引不同深度的枚举集合。考虑到java的回收机制，我们直接用新的枚举集合替代旧的，以使得旧的集合占据的资源可以被释放和回收。

我们对论文提出算法的若干循环做了最大限度的合并，以提高效率。

# 实验结果

 实验在给定的三种数据分布（inde、corr、anti）下4种维度（2、4、6、8）共12组数据上进行。实验的group size k取值分别为2、4、6。

实验环境：

## 3.1生成DSG实验结果

图表1、图表2、图表3分别展示了k取2、4、6时在上述12种数据集上生成skyline layers并进行预处理构造DSG的耗时。在维度为2时采取二分查找的方式进行插入，而在高维度时采取简单的遍历方式。通过图表可以发现，在k取不同值、在具有不同分布的数据集上生成skyline layers和构造DSG时，维度为2时的耗时要明显小于高维情况，也就是说，论文中针对维度为2时的二分处理具有较好的性能优势。

图表1

图表2

图表3

## 3.2两种生成G-skyline的算法实验结果