超市数据的频繁项集挖掘 – b问分析文档

数据分析与数据挖掘

DAM COURSE, SPRING 2018

BY

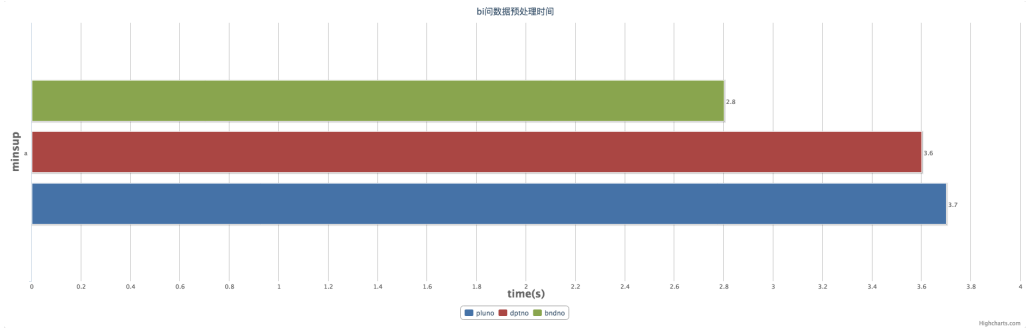
1552674 李　源

Tongji University
School of Software Engineering

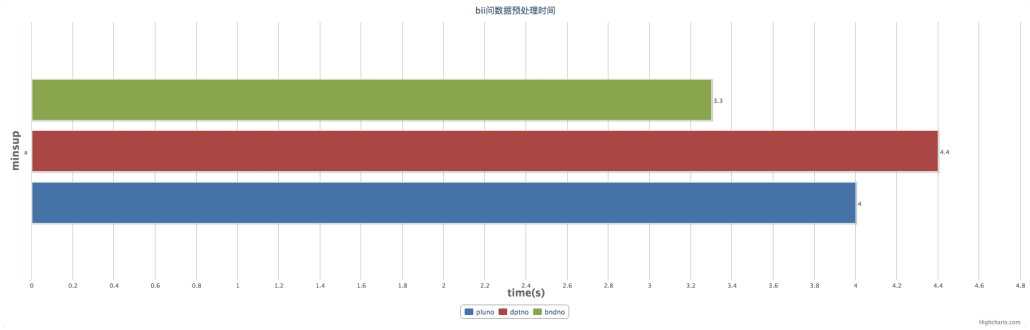
1 trade_new.csv文件数据代码运行结果

1.1 数据预处理耗时

针对bi问的数据预处理时间如下：



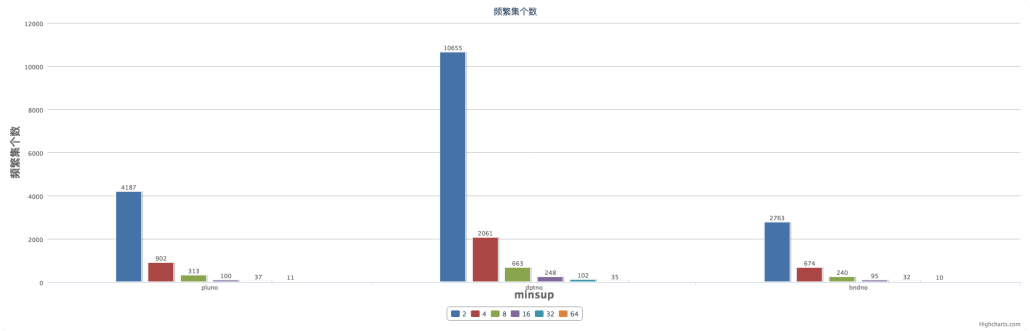
针对bii问的数据预处理时间如下：



1.2 bi问针对三个字段的频繁集求取情况

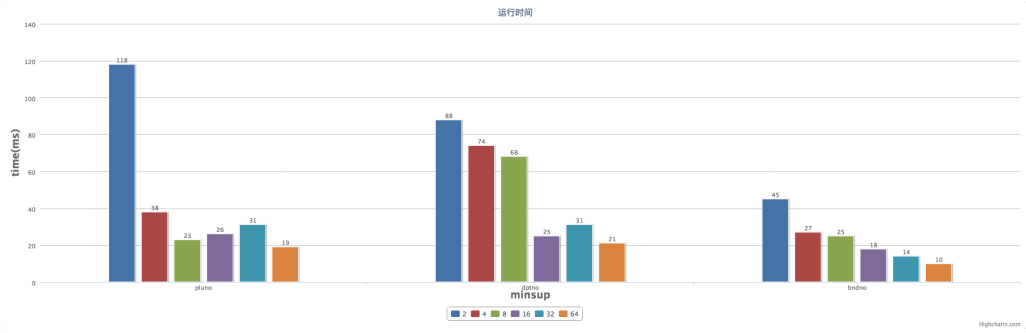
1.2.1 频繁集个数

针对pluno、dptno、bndno字段，在2、4、8、16、32、64分别作为最小支持度时，频繁集个数：



1.2.2 运行时间

针对pluno、dptno、bndno字段，在2、4、8、16、32、64分别作为最小支持度时，运行时间：



1.2.3 具体输出

针对pluno、dptno、bndno字段，在16作为最小支持度时，使用SPMF包获得的输出结果：
pluno字段：

```
Algorithm is running...
==========  PREFIXSPAN 0.99-2016 - STATISTICS ==========
 Total time ~ 26 ms
 Frequent sequences count : 100
 Max memory (mb) : 265.86871337890625
 minsup = 16 sequences.
 Pattern count : 100
=============================================================
```

dptno字段：

```
Algorithm is running...
==========  PREFIXSPAN 0.99-2016 - STATISTICS ==========
 Total time ~ 30 ms
 Frequent sequences count : 248
 Max memory (mb) : 265.86871337890625
 minsup = 16 sequences.
 Pattern count : 248
=============================================================
```

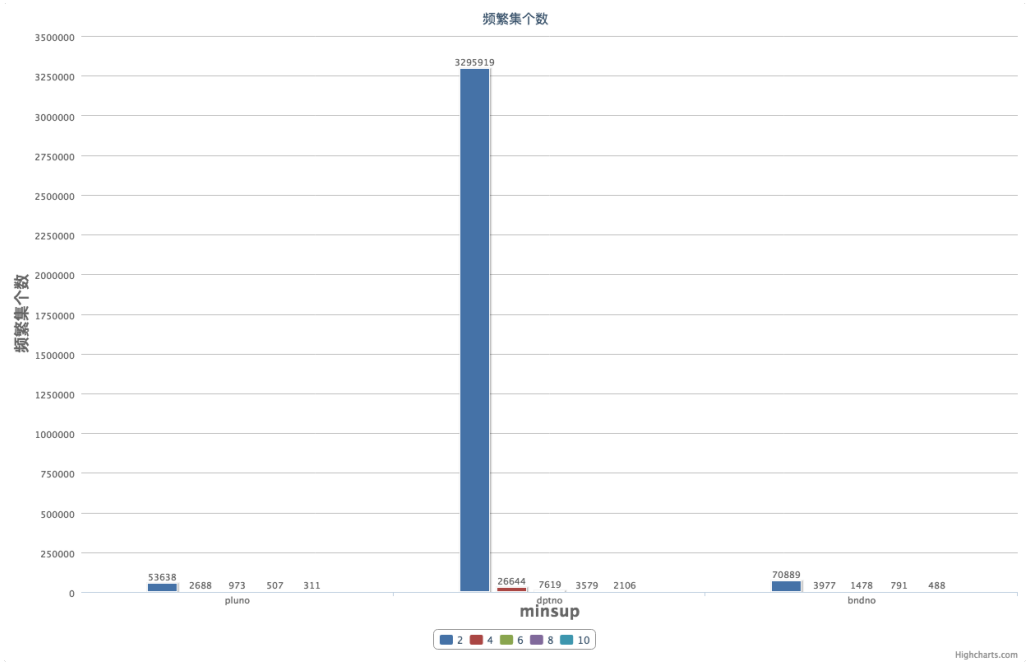
bndno字段：

```
Algorithm is running...
==========  PREFIXSPAN 0.99-2016 - STATISTICS ==========
 Total time ~ 18 ms
 Frequent sequences count : 95
 Max memory (mb) : 265.86871337890625
 minsup = 16 sequences.
 Pattern count : 95
=============================================================
```

1.3 bii针对三个字段的频繁集求取情况

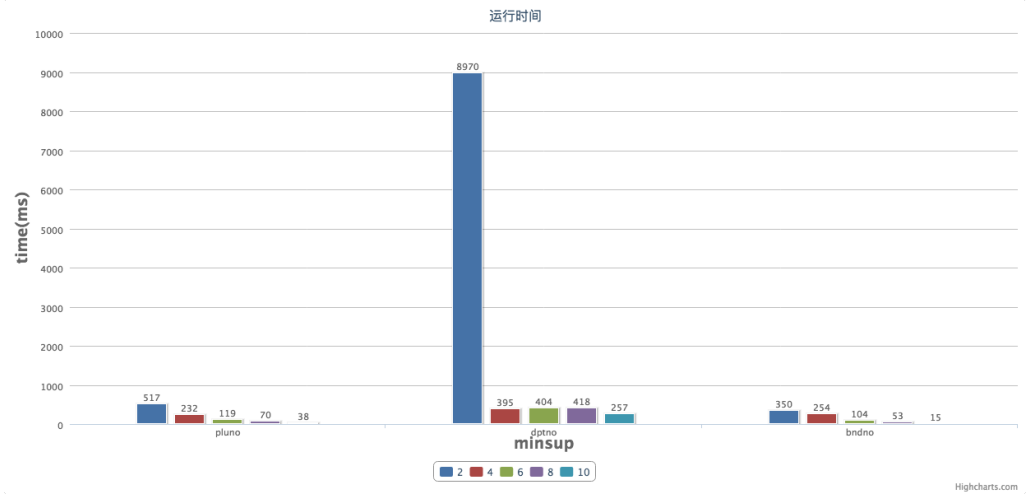
1.3.1 频繁集个数

针对pluno、dptno、bndno字段，在2、4、6、8、10分别作为最小支持度时，频繁集个数：



1.3.2 运行时间

针对pluno、dptno、bndno字段，在2、4、6、8、10分别作为最小支持度时，运行时间：

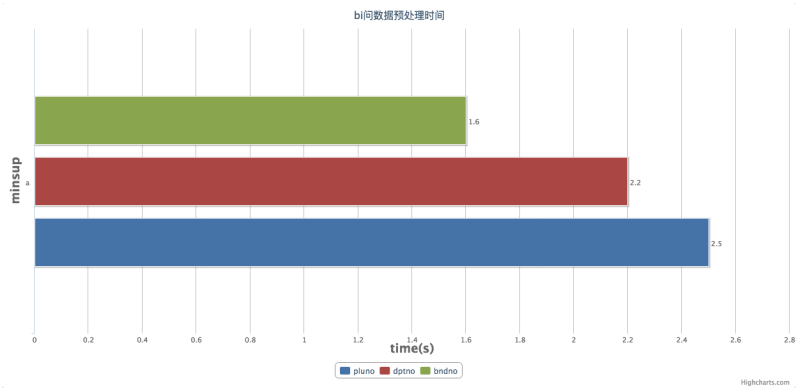


我在这里选择的是PrefixSpan算法。

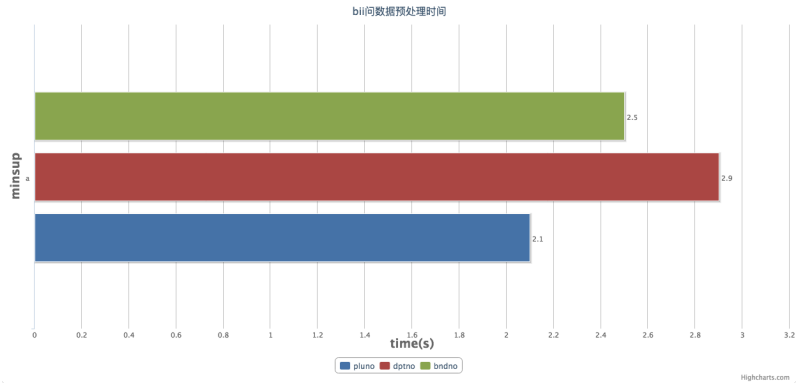
总的来说，除了对应dptno字段在最小支持度为2的时候，所耗时间较长之外，其他的运行时间都比较短，同时与a问的FP-growth算法相比，基本上差异不大，可见两者的性能差异不大。而对于频繁项集的结果，我将会在分析讨论中做一个详细说明。

2 trade.csv文件数据代码运行结果

针对bi问的数据预处理时间如下：



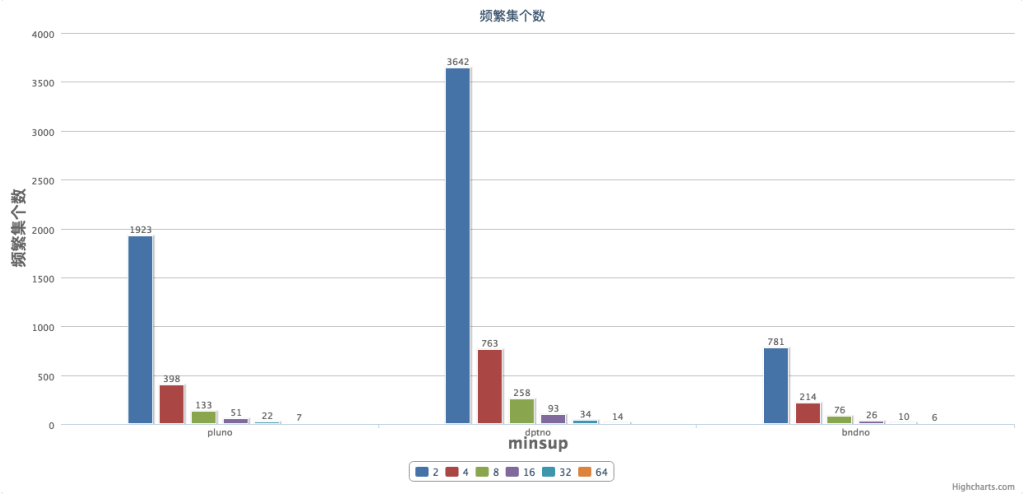
针对bii问的数据预处理时间如下：



2.1 bi问针对三个字段的频繁集求取情况

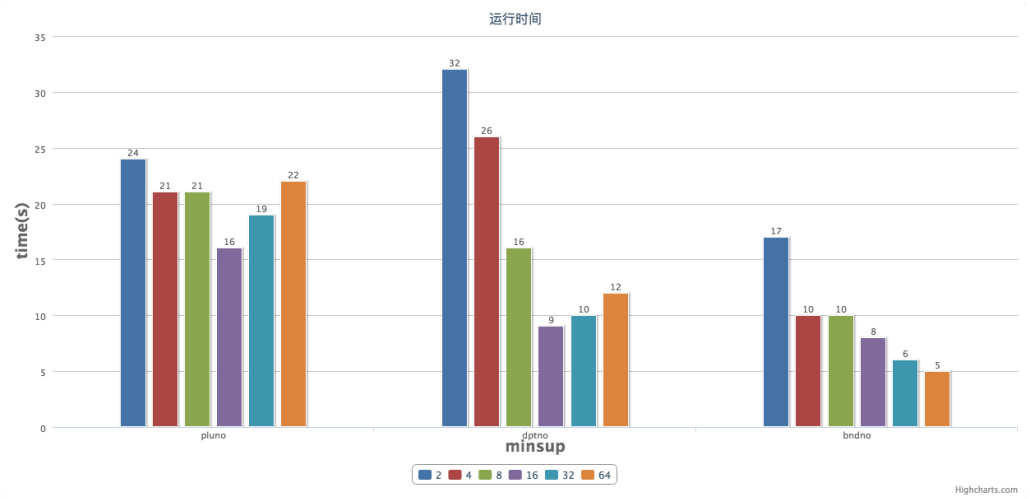
2.1.1 频繁集个数

针对pluno、dptno、bndno字段，在2、4、8、16、32、64分别作为最小支持度时，频繁集个数：



2.1.2 运行时间

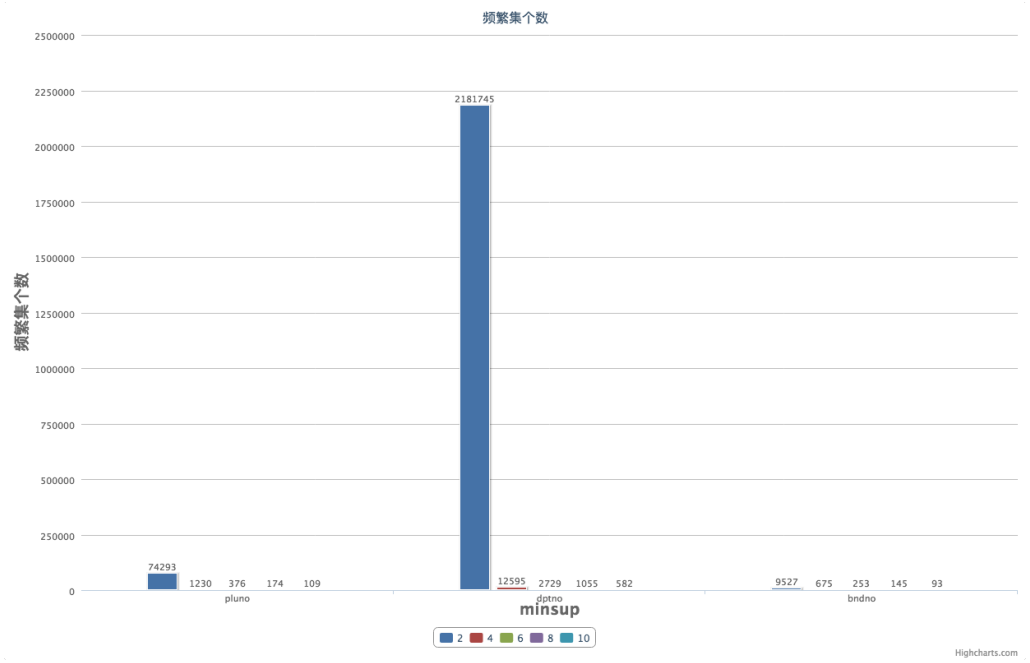
针对pluno、dptno、bndno字段，在2、4、8、16、32、64分别作为最小支持度时，运行时间：



2.2 bii问针对三个字段的频繁集求取情况

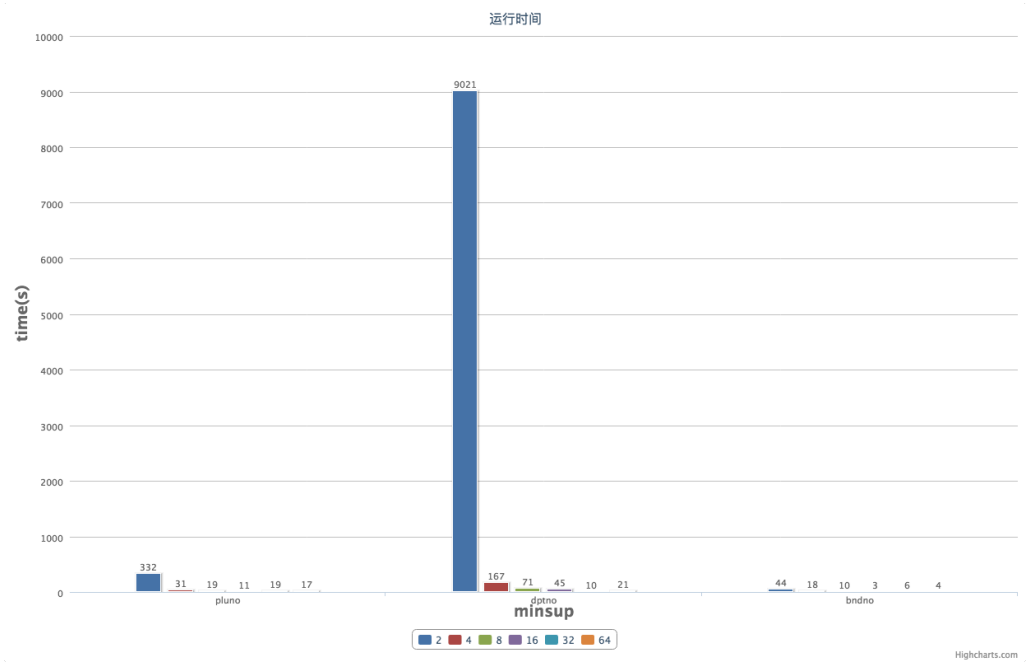
2.2.1 频繁集个数

针对pluno、dptno、bndno字段，在2、4、6、8、10分别作为最小支持度时，频繁集个数：



2.2.2 运行时间

针对pluno、dptno、bndno字段，在2、4、6、8、10分别作为最小支持度时，运行时间：



3 分析讨论

这里主要针对数据的预处理和ai、bi问的差异性来进行分析讨论。

3.1 数据的预处理

b问和a问的最大一点不同在于，b问需要考虑时序性，体现在本问当中的话，我是考虑将时间相同的数据合并作为一个值来考虑，也就是合并为$< a(ab)c >$的格式。

这里我的处理方法与a问的不同主要体现在最后一步合并上，大致的步骤如下：

1）仍然用字典来做存储，不过是两层字典。第一层的key为vipno值，value为一个字典；而第二层的这个字典，key为sldat，即时间，value为具体的item_no值。

这样最终可以能够将时间相同的数据合并在了一起。

3.2 a、b问的差异性

首先说ai问和bi问。因为这两问都是针对uid去进行的数据合并，而对于同一个uid，其购买时间肯定是一样的，其合并后的结果和ai问是一样的，那么其实ai和bi的输入数据并没有太大的差异。我在这里选择出了支持度前十的针对pluno字段的频繁项集，其结果如下：

```
[['30380003'], 415]
[['30380002'], 223]
[['23110009'], 175]
[['22036000'], 117]
[['27410000'], 84]
[['27000582'], 81]
[['22102014'], 79]
[['27300274'], 77]
[['23110001'], 76]
[['25120016'], 72]
```

Figure 3.1: ai问

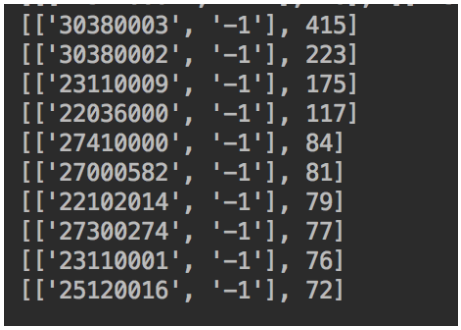


Figure 3.2: bi问（-1是分隔符）

显然，结果是一样的，这也符合我们的推测。

那么对于aii问和bii问，会不会有所不同呢？我同样是选择出了支持度前十的针对pluno字段的频繁项集，其结果如下：

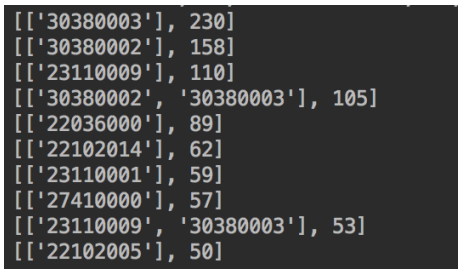


Figure 3.3: ai问

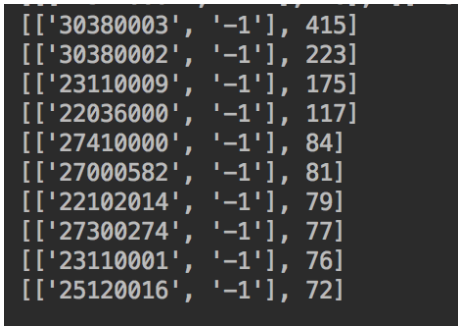


Figure 3.4: bi问（-1是分隔符）

显然，结果出现了一定的差异，这是符合常理的，毕竟两种算法所运用到的原理存在着一定的差异性。与FP-growth算法不同的是，PrefixSpan算法引入了前缀、后缀的概念，即我们可以简单地理解为，出现在a后面的数据为a的后缀，而a就为这些数据的前缀。

那么就会出现这样一个情况，比如有$< a(bc) >$、$< (ab) >$这么两组数据，那么我们在计算a的后缀时，b要写作b、$_b$分别考虑，因为前者和a购买存在时间间隔，而后者则时间间隔在允许范围内，或者就是是同一次购买。这也是对应于FP-growth算法的一个很大的区别。

但是我发现支持度前十的频繁项集，还是存在着一定的相似度的，比如3038003、3038002都是排着前二的，而且其他频繁项集中对应的项也存在着一定的相似度。

那么我可以这样认为，对于本次作业的数据，虽然是否考虑时序性会对最终的频繁项集产生一定的影响，但是总体来说，本次作业的数据即超市购买信息和时间关系不大，也就是说，这些购买信息对应的用户的购买喜好并没有因为时间产生很大的差异性。当然至少对于本份数据是这样的。

4 性能比较

我们可以查看FP-growth和PrefixSpan算法两者的运行时间如下：

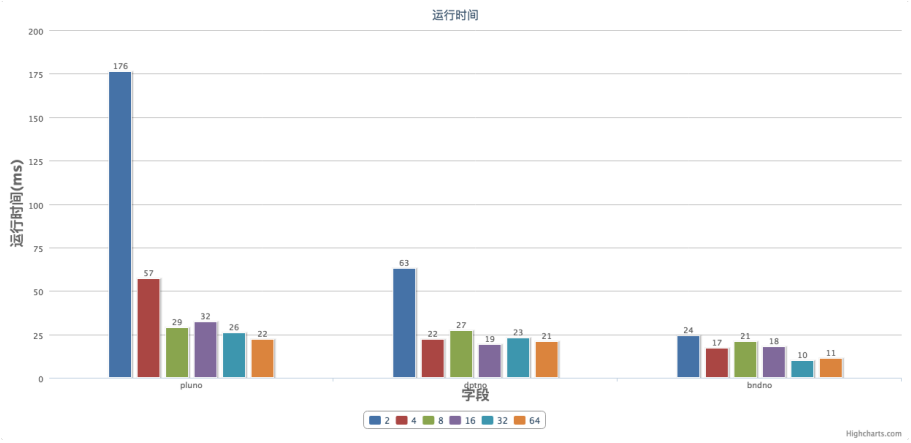


Figure 4.1: FP-growth

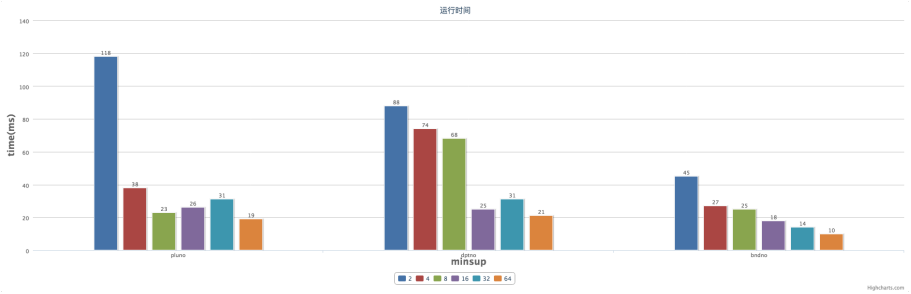


Figure 4.2: PrefixSpan

总的来说差异不大，FP-growth会稍微快一点。考虑到本次数据大约有万级左右，并不算特别大，我可以粗略地认为这两种算法之间的性能相差不大。