# PrefixTreeESpan 算法实验报告

扈煊 1201214133

2012 年 11 月 20 日

# 1 实验目的

- 理解并实现基于模式增长的频繁子树挖掘算法 PrefixTreeESpan [1]

  （**Prefix-Tree**-projected **E**mbedded-**S**ubtree **pa**ttern**n**）

- 验证算法基本正确性并进行可能的优化和改进

# 2 作业说明

## 2.1 目录文件结构

本次作业打包包含的内容及其说明如下：

```
PrefixTreeESpan
├── main.py ..... 算法实现主程序
├── tree.py ..... 辅助类 tree.Node
├── project.py ..... 辅助类 project.Project
├── README.txt ..... 简要说明
├── test.in ..... demo 测试文件
├── Makefile ..... Makefile 脚本，用于 make 方式运行命令
├── data/ ..... 数据集目录
│   ├── CSlog.data
│   ├── D10.data
│   ├── F5.data
│   └── T1M.data
├── output/ ..... 输出结果目录，对应不同的测试编号
│   ├── CSlog-1.out
│   ├── D10-1.out
│   ├── D10-2.out
│   ├── D10-3.out
│   ├── F5-1.out
│   ├── F5-2.out
│   └── F5-3.out
└── report/ ..... 实验报告目录
    ├── report.tex ..... LaTeX 源文件
    ├── report.bib ..... BiBTeX 参考文献文件
    └── Makefile ..... TeXmake 脚本
```

## 2.2 运行方法

### 2.2.1 Makefile

Makefile 脚本的运行方法正对特定的数据集，需要支持 make 命令的环境，即??节实验结果中对应的不同测试样例，运行方法为"make 测试编号"，如测试编号为“CSlog-1”，则运行命令为“make CSlog-1”，具体测试编号请见表2。

### 2.2.2 Python

Python 脚本的运行方法针对通用的数据集，可以自定义输入文件、输出文件和最小支持度

**命令行** python main.py -i inputfile -o outputfile -s minsup

**参数说明**

**-i** 必选，输入文件，即图数据库对应文件，默认值为”test.in”，为论文中提到的实例

**-o** 可选，输出文件，即结果输出文件，默认值为”test.out”

**-s** 必选，最小支持度，整型，默认值为”2”

# 3 算法设计

## 3.1 主要思想

- 频繁子树的推导子树一定是频繁的
- 频繁子树总是可以通过其前缀树增长得到
- 利用深度优先搜索的思想通过递归迭代不断对前缀树进行增长，统计频繁的增长因子，从而得到更多的频繁子树

## 3.2 伪代码

**Algorithm 1:** PrefixTreeESpan()

**Data**: 树数据库 $TreeDB$, 最小支持计数 $MINSUP$

利用栈深度优先遍历每棵树中所有节点，计算配对范围 ;

利用集合合并每棵树中的标签，统计标签计数 ;

将每个标签计数与最小支持计数比较得一阶频繁标签集合 ;

**for** 一阶频繁子树 $in$ 一阶频繁子树集合 **do**

    保存和输出一阶频繁子树即前缀树 $pre\_tree$ ;

    **for** 一阶频繁子树在数据库 $TreeDB$ 中的出现 **do**

        对于每一例出现生成其对应的投影实例 ;

        将投影实例加入到新的投影数据库中 $ProDB$ ;

    调用函数 $get\_fre$ ( $pre\_tree$, 1, $ProDB$ ) ;

**Algorithm 2:** $get\_fre$ ( $pre\_tree$, $n$, $ProDB$ )

利用集合统计所有投影实例中的增长因子计数 ;

将每个增长因子计数与最小支持计数比较得到频繁增长因子 ;

**for** $n+1$ 阶增长因子 $in$ $n+1$ 阶增长因子集合 **do**

    生成、保存并输出 $n+1$ 阶频繁子树即前缀树 $pre\_tree\_new$ ;

    **for** $n+1$ 阶增长因子在投影数据库 $ProDB$ 中的出现 **do**

        对于每一例出现生成其对应的投影实例 ;

        将投影实例加入到心的投影数据库中 $ProDB\_new$ ;

    调用函数 $get\_fre$ ( $pre\_tree\_new$, $n+1$, $ProDB\_new$ ) ;

# 4 算法实现

## 4.1 算法实现概述

本次实验使用了 Python 语言进行了代码的编写，采用 Python 语言的主要原因除了个人使用比较熟练以外，还考虑到可以使用 Python 中的高级数据结构，如列表、字典和集合，来简化一些处理。并且在处理命令行参数和字符串处理等方面，Python 包装的很好使用起来更顺手一些。

## 4.2 辅助类

**tree.Node** 表示树中的每一个节点，记录其标号和匹配范围

**Node.label** 表示该节点的标号

**Node.label** 表示该节点对应的匹配位置范围，即与其对应的”-1” 的序号

**project.Project** 封装为一个投影数据库的信息，主要参考了论文中所述的伪投影法，为了方便使用有所修改。

**Project.tid** 表示该投影数据库所在的树的序号

**Project.offset_list** 表示该投影数据库每一部分在树中的偏移量列表

**Project.scope_list** 表示该投影数据库每一部分在树中每一个偏移量对应父节点的匹配位置范围列表

## 4.3 伪投影数据库

论文原文中的伪投影数据库采用的是三元组的形式，但是不难发现 tid，即树的编号是重复的，因为投影数据库只可能是某一棵树的子树或森林，而不可能跨越不同的树，所以本次实现中将投影数据库封装成了一个类，其中 tid 表示该树的序号，三元组中的其他两个元素分别用列表对应存储。具体实现见代码2。

# 5 实验结果

## 5.1 实验环境

**CPU** Intel® Core™2 Duo CPU T7500 @ 2.20GHz × 2

**RAM** 1G * 2

**OS** Ubuntu-Desktop 12.10 64-bit

## 5.2 测试数据

本次测试选用了老师提供的 4 个测试集，每个测试集的文件大小及其包含树的个数见表1

表 1: 数据集文件大小及其包含树的个数

| 数据集 | 文件大小 (M) | 树的个数 |
|---|---|---|
| CSlog | 6.0 | 59691 |
| D10 | 1.7 | 100000 |
| F5 | 1.8 | 100000 |
| T1M | 15.0 | 1000000 |

## 5.3 实验结果

本次实验主要针对5.2节中提到的数据集进行了测试，每个数据集分别选用了最小支持度百分比为 10%、1% 和 0.1% 三个比例进行了测试，所有输出结果都在 output 文件夹中，对应的文件名为“测试编号.out”，如测试编号 D10-1 对应的输出文件为“D10-1.out”，输出内容为所有挖掘出来的频繁子树，最后一行为所消耗的时间，得到的统计结果见表2。

表 2: 实验结果

| 数据集 | 测试编号 | MINSUP 个数 | MINSUP 百分比 | 频繁子树个数 | 运行时间 |
|---|---|---|---|---|---|
| CSlog | CSlog-1 | 5969 | 10 | 2 | 00:06.95 |
| D10 | D10-1 | 10000 | 10 | 6 | 00:04.17 |
| D10 | D10-2 | 1000 | 1 | 405 | 00:22.13 |
| D10 | D10-3 | 100 | 0.1 | 7426 | 00:59.79 |
| F5 | F5-1 | 10000 | 10 | 16 | 00:06.11 |
| F5 | F5-2 | 1000 | 1 | 180 | 00:15.30 |
| F5 | F5-3 | 100 | 0.1 | 928 | 00:25.72 |

# 6 实验总结

## 6.1 关于 T1M

比较实验数据和实验结果，不难看出没有 T1M 数据集相关的结论，在试验过程中运行 T1M 数据集时，内存占用会迅速膨胀，不用一会就会将整个计算机的内存耗尽，所以就没有让程序继续运行。CSlog 数据集的最小支持度为 1% 和 0.1% 的测试用例也是同样的原因没有得到最终结果。主要原因应该有以下几个方面：

- Python 作为解释型脚本语言，本身运行时就会占用较多内存，对内存的要求也比较高
- 算法中使用了递归的逻辑，没有进行非递归实现
- 算法实现过程中使用了较多高级数据结构，尤其在递归中还作为了参数，用来当前的前缀树和投影数据库

## 6.2 关于效率

出本次算法实现的效率和论文中已经实现的方法相比总体偏低，个人觉得主要原因有如下几点：

- Python 语言本身执行效率没有 C/C++ 之类的语言高
- 未实现论文中已提到的另一个优化，提前减枝，删去数据中不频繁的节点
- 算法实现的其他细节还没有充分优化，比如文件输出、内存清空等

# 7 源代码及注释

```
"""
File: tree.py
Author: huxuan
E-mail: i(at)huxuan.org
Created: 2012-11-13
Last modified: 2012-11-14
Description:
    Assistant Class for Tree

Copyrgiht (c) 2012 by huxuan. All rights reserved.
License GPLv3
"""

class Node(object):
    """Node in a tree

    Attributes:
        label: the label of the node
        scope: scope range in string represented tree
    """
    def __init__(self, label=""):
        """Init Node"""
        self.label = label
        self.scope = 0
```

代码 1: 辅助类 tree.Node

```
"""
File: project.py
Author: huxuan
E-mail: i(at)huxuan.org
Created: 2012-11-13
Last modified: 2012-11-14
Description:
```

```
    Assist Classes for Projection Database

Copyrgiht (c) 2012 by huxuan. All rights reserved.
License GPLv3
"""

class Project(object):
    """Summary of Project

    Attributes:
        tid: id of the tree
        offset_list: offset for each subset of projection database
        scope_list: corresponding scope for each offset
    """
    def __init__(self, tid):
        """Init Project"""
        self.tid = tid
        self.offset_list = []
        self.scope_list = []

    def add(self, offset, scope):
        """Add a ProjectItem in to list"""
        self.offset_list.append(offset)
        self.scope_list.append(scope)

    def __str__(self):
        """Redefine the output of Project"""
        return "Tid: %d %s %s" % (self.tid, self.offset_list, self.
            scope_list)
```

代码 2: 辅助类 project.Project

```
"""
File: main.py
Author: huxuan
```

```
4 E-mail: i(at)huxuan.org
5 Created: 2012-11-12
6 Last modified: 2012-11-14
7 Description:
8     Implementation of PrefixTreeESpan
9     A Frequent Subtree Mining Algorithm
10
11 Copyrgiht (c) 2012 by huxuan. All rights reserved.
12 License GPLv3
13 """
14
15 import os
16 import sys
17 import getopt
18 import datetime
19
20 from tree import Node
21 from project import Project
22
23 TREE_DB = []
24 MINSUP = 0
25 TREEDATA = 'in.in'
26 OUTPUT = 'test.out'
27
28 def init():
29     """
30     Global Initialization
31     """
32
33     # Check exist of output file, remove if exists
34     if os.path.exists(OUTPUT):
35         os.remove(OUTPUT)
36
37 def get_opt():
```

```
38      """
39      Handle the options
40      """
41
42      global MINSUP
43      global TREEDATA
44      global OUTPUT
45
46      # Parse options and arguments
47      optlist, args = getopt.getopt(sys.argv[1:], 'i:s:o:')
48
49      # Deal with predefined options
50      for opt, arg in optlist:
51          if opt == '-i':
52              TREEDATA = arg
53          elif opt == '-s':
54              MINSUP = int(arg)
55          elif opt == '-o':
56              OUTPUT = arg
57          else:
58              continue
59
60      return 0
61
62  def output_pre_tree(pre_tree):
63      """
64      Append result to output file
65      """
66      output_file = file(OUTPUT, 'a+')
67      print >> output_file, ' '.join(pre_tree)
68      output_file.close()
69
70  def get_fre(pre_tree, n, pros_db):
71      """
```

```
72      PrefixTreeESpan Algorithm (Part 2)
73      Get n+1 order frequent subtree according to
74      prefix tree and projection database
75      """
76
77      # Count growth elements
78      growth_elems_count = {}
79      for pros in pros_db:
80          tree = TREE_DB[pros.tid]
81          # Traversal each project database
82          for i in xrange(len(pros.scope_list)):
83              for j in xrange(pros.offset_list[i], pros.scope_list[i])
                    :
84                  if tree[j].label != '-1':
85                      growth_elem = (tree[j].label, i + 1)
86                      # Treat the count as set of tree id
87                      if growth_elem not in growth_elems_count:
88                          growth_elems_count[growth_elem] = set([])
89                      growth_elems_count[growth_elem].add(pros.tid)
90
91      # print '=' * 80
92      # print "F%d Growth Elements Counts:" % (n + 1, )
93      # for growth_elem in growth_elems_count:
94      #     print growth_elem, len(growth_elems_count[growth_elem])
95
96      # Get frequent growth elements via comparison with MINSUP
97      fre_elems = set([ growth_elem
98          for growth_elem in growth_elems_count
99              if len(growth_elems_count[growth_elem]) >= MINSUP ])
100
101     # print '=' * 80
102     # print "F%d Frequent Elements:" % (n + 1, )
103     # print fre_elems
104
```

```
105     # Get projection database for each frequent growth elements
106     for fre_elem in fre_elems:
107 
108         # print '=' * 80
109         # print "F%d Frequent Element:" % (n + 1, )
110         # print fre_elem
111 
112         # Generate new prefix tree
113         pre_tree_new = pre_tree[:]
114         pre_tree_new.insert(- fre_elem[1], fre_elem[0])
115         pre_tree_new.insert(- fre_elem[1], '-1')
116         # Output the result
117         output_pre_tree(pre_tree_new)
118 
119         # Generate new projection database
120         pros_db_new = []
121         for i in xrange(len(pros_db)):
122             pros = pros_db[i]
123             tree = TREE_DB[pros.tid]
124             for j in xrange(len(pros.offset_list)):
125                 for k in xrange(pros.offset_list[j], pros.scope_list[
                        j]):
126                     if tree[k].label == fre_elem[0]:
127                         pros_new = Project(pros.tid)
128                         if tree[k + 1].label != '-1':
129                             pros_new.add(k + 1, tree[k].scope)
130                         l = tree[k].scope + 1
131                         while tree[l].label != "-1" and l < pros.
                                scope_list[j]:
132                             pros_new.add(l, tree[l].scope)
133                             l = tree[l].scope + 1
134                         pros_db_new.append(pros_new)
135 
136         # print '=' * 80
```

```
137         # print 'F%d Projected Database:' % (n + 1, )
138         # for pros in pros_db_new:
139         #     print pros
140 
141         # Get next level frequent subtree
142         get_fre(pre_tree_new, n + 1, pros_db_new)
143 
144     return 0
145 
146 def prefixtreeespan():
147     """
148     PrefixTreeESpan algorithm (Part 1)
149     Get first level frequent nodes and corresponding projection
            database
150     """
151     # Read date file, compute useful params
152     growth_elems_count = {}
153     tree_data = file(TREEDATA)
154     for tree_string in tree_data.readlines():
155 
156         # Get label list
157         tree_labels = tree_string.strip().split(' ')
158         # print "Labels:", tree_labels
159 
160         # DFS to compute scope
161         root = Node(tree_labels[0])
162         stack = [(root, 0)]
163         tree = [root]
164         index = 1
165         while stack:
166             elem = Node(tree_labels[index])
167             if elem.label == '-1':
168                 stack[-1][0].scope = index
169                 stack.pop()
```

```
170             else:
171                 stack.append((elem, index))
172             tree.append(elem)
173             index += 1
174 
175         # Add tree in to global TREE_DB
176         TREE_DB.append(tree)
177 
178         # print '=' * 80
179         # print "F1 Elements:"
180         # for elem in tree:
181         #     print tree.index(elem), elem.label, elem.scope
182 
183         # Count all node as growth elements in set
184         growth_elems = set([ (elem.label, 0)
185             for elem in tree
186                 if elem.label != '-1' ])
187         for growth_elem in growth_elems:
188             if growth_elem not in growth_elems_count:
189                 growth_elems_count[growth_elem] = 0
190             growth_elems_count[growth_elem] += 1
191 
192     # print '=' * 80
193     # print "F1 Growth Elements Count:"
194     # for growth_elem in growth_elems_count:
195     # print growth_elem, growth_elems_count[growth_elem]
196 
197     # Get frequent one level nodes
198     fre_elems = set([ growth_elem
199         for growth_elem in growth_elems_count
200             if growth_elems_count[growth_elem] >= MINSUP ])
201 
202     # print '=' * 80
203     # print "F1 Frequent Elements:"
```

```
204     # print fre_elems
205
206     # Generate projection dateabase for each frequent node
207     for fre_elem in fre_elems:
208
209         # print '=' * 80
210         # print "F1 Frequent Element:"
211         # print fre_elem
212
213         pre_tree = [fre_elem[0], '-1']
214         output_pre_tree(pre_tree)
215
216         pros_db = []
217         for i in xrange(len(TREE_DB)):
218             tree = TREE_DB[i]
219             for j in xrange(len(tree)):
220                 if tree[j].label == fre_elem[0] and tree[j + 1].label
                        != '-1':
221                     pros = Project(i)
222                     pros.add(j + 1, tree[j].scope)
223                     pros_db.append(pros)
224
225         # print '=' * 80
226         # print 'F1 Projected Database:'
227         # for pros in pros_db:
228         #     print pros
229
230         get_fre(pre_tree, 1, pros_db)
231
232 def main():
233     """
234     Main process
235     """
236     # Get Options for input/output file and minsup
```

```
237     get_opt()
238
239     # Some Initilization
240     init()
241
242     # Mark start time for calculating time consuming
243     time_start = datetime.datetime.now()
244
245     # Call prefixtreeespan algorithm
246     prefixtreeespan()
247
248     # Mark end time and calculate time consuming
249     time_end = datetime.datetime.now()
250     output_file = file(OUTPUT, 'a+')
251     print >> output_file, "Time used: ", time_end - time_start
252     output_file.close()
253
254 if __name__ == '__main__':
255     main()
```

代码 3: PrefixTreeESpan 算法实现主体

# 参考文献

[1] Lei Zou, Yansheng Lu, Huaming Zhang, and Rong Hu. Prefixtreeespan: A pattern growth algorithm for mining embedded subtrees. In *Proceedings of the 7th international conference on Web Information Systems*, WISE'06, pages 499–505, Wuhan, China, 2006. Springer-Verlag.