文章编号:1001 - 9081(2011)05 - 01391 - 04
doi:10.3724/SP.J.1087.2011.01391

无项头表的 FP-Growth 算法

凌绪雄[1,2],王社国[1],李　洋[2,3],苗再良[2]
(1. 河北工程大学 信息与电气工程学院,河北 邯郸 056000;
2. 浪潮集团博士后工作站,济南 250101;　3. 山东大学 博士后流动站,济南 250101)
(lingxuxiong@foxmail.com)

摘　要:针对 FP-Growth 算法中频繁模式树的遍历低效问题,提出了一种无项头表的频繁模式增长算法。该算法利用递归回溯的方式遍历频繁模式树以求取条件模式基,解决了对同一树路径多次重复遍历的问题。从理论分析和实际挖掘能力两方面,将新算法与 FP-Growth 算法进行了对比。结果表明,新算法有效减少了条件模式基的搜索开销,使频繁模式挖掘的效率提高了 2 ~ 5 倍,在时间和空间性能上均优于 FP-Growth 算法。将该算法应用于通信告警关联规则挖掘,较快地挖掘出了关联规则结果,且正确规则的覆盖率达到了 83.3%。

关键词:项头表;频繁模式;关联规则;告警关联;数据挖掘
中图分类号: TP311.13　　**文献标志码**:A

No-header-table FP-Growth algorithm

LING Xu-xiong[1,2], WANG She-guo[1], LI Yang[2,3], MIAO Zai-liang[2]
(1. *College of Information and Electrical Engineering, Hebei University of Engineering, Handan Hebei* 056000, *China*;
2. *INSPUR Post Doctors Work Station, Jinan Shandong* 250101, *China*;
3. *Post-doctoral Station, Shandong University, Jinan Shandong* 250101, *China*)

Abstract: Concerning the problem of low traversal efficiency when searching the FP-tree for conditional pattern bases, a new no-header-table FP-Growth algorithm was proposed. The algorithm employed a recursively backtracking way to search for conditional pattern bases, avoiding traversing the same FP-tree path multiple times. Compared with FP-Growth algorithm in terms of theoretical analysis and actual mining performance, this algorithm greatly reduced the searching cost and improved the mining efficiency of frequent patterns by 2 - 5 times. Finally, the algorithm was used to mine association rules in telecommunication network alarms. The high mining speed, with the coverage of 83.3% against correct rules, shows that it is superior to FP-Growth both in time and space performance.

Key words: item header table; frequent pattern; association rule; alarm association; data mining

0　引言

频繁模式挖掘是数据挖掘领域中的一个重要分支,其目标是从特定的事务数据库中找出满足最小支持度阈值的频繁模式,进而导出有意义的关联规则,已在通信网络、生命科学等领域获得了应用[1-3]。频繁模式搜索算法包括传统的基于枚举-测试思想的 Apriori 算法[4-5]和基于频繁模式树的 FP-Growth 算法[6-8]。Apriori 算法枚举候选项集和多次扫描数据库会消耗大量的时间,适合于挖掘短模式。FP-Growth 算法将事务压缩到紧凑的树结构中,利用动态规划的思想,将挖掘全部频繁模式的问题转化为递归挖掘以特定后缀结尾的系列子频繁模式的问题,从而使挖掘速度有了一个数量级的提升[6]。但 FP-Growth 算法存在对同一树路径的多次重复遍历问题[9]。当数据量越大,尤其是事务的平均长度越长,事务的压缩度越高时,其挖掘效率就越低。文献[9]通过添加遍历标志来避免对同一路径的重复搜索,但其时间复杂度仍为 $O(N^2)$,且增加条件标志造成了存储空间的浪费。本文充分利用路径共享的优势,通过递归遍历频繁模式树的方式,使算法的时间复杂度降低为 $O(N)$,较好地解决了同一路径的多次重复遍历问题,同时避免了存储空间的额外开销。最后,将改进后的算法应用于通信网络告警关联规则挖掘,在较短的时间内挖掘出大量有意义的关联规则,能够满足通信告警关联规则挖掘的需要。

1　无项头表的频繁模式增长算法

1.1　频繁模式树公共路径的重复遍历问题

传统的 FP-Growth 算法按项头表自底向上的顺序,依次进行求取条件模式基、建立条件模式树的递归挖掘。在整个递归挖掘频繁模式的过程中,会对同一树路径造成多次重复遍历,影响了算法的性能。考虑如图 1 所示的频繁模式树。

搜索 m 的条件模式基时,需根据同项指针自底向上搜索{a -> c -> f -> R:2, b -> a -> c -> f -> R: 1}两条路径。整个递归挖掘过程中,全部频繁一项的搜索路径如表 1 所示。由表 1 可知,在搜索全部条件模式基的整个过程中,单〈a -> c -> f -> R〉路径就被重复搜索了 4 次。通过进一步的观察发现,被重复遍历的路径〈a -> c -> f -> R〉恰好是树的最左边两个分支的公共路径。由此可见,事务的压缩程度越高,频繁一项集的长度越长,公共路径被重复搜索的概率就越大。

收稿日期:2010 - 12 - 01;**修回日期**:2011 - 01 - 19。

作者简介:凌绪雄(1985 -),男,四川凉山人,硕士,主要研究方向:数据挖掘、语音识别;　王社国(1967 -),男,河北永年人,副教授,主要研究方向:语音识别、图像处理、数据库;　李 洋(1978 -),男,黑龙江哈尔滨人,博士,主要研究方向:神经网络、专家系统、数据挖掘;　苗再良(1961 -),男,山东潍坊人,高级研究员,主要研究方向:智能化移动互联网、告警监控。

为充分利用共享路径的优势，提出了基于递归回溯的无项头表的频繁模式增长算法（No Header Table Fast Pattern Growth algorithm，NHTFPG）。该算法只需遍历一次频繁模式树即可找出所有频繁一项的条件模式基，从而避免了对公共路径的重复遍历。

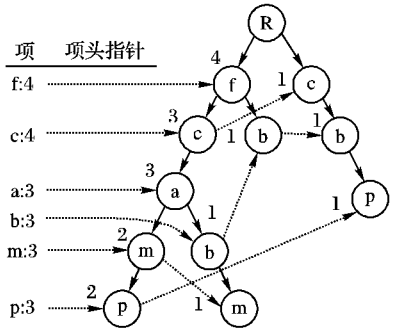

图 1 频繁模式树结构示意

表 1 频繁一项及其条件模式基

频繁一项	条件模式基
p	〈m ->a ->c ->f ->R:2,b ->c ->R:1〉
m	〈a ->c ->f ->R:2,b ->a ->c ->f ->R:1〉
b	〈a ->c ->f ->R:1, f ->R:1, c ->R:1〉
a	〈c ->f ->R:3〉
c	〈f ->R:3〉
f	〈R〉

1.2 无项头表的快速条件模式基搜索

由 FP-tree 的建树过程知，具有共同前缀的事务项被压缩到相同的 FP-tree 分支中，仅当当前插入项跟已有路径项不同时，插入路径才出现分叉，产生新的分支路径。为说明方便，作如下定义。

定义 1 分叉节点，孩子数大于 1 的树节点，简称 F 节点。单支节点，孩子数为 1 的树节点，简称 S 节点。叶子节点，孩子为空的树节点，简称 L 节点。

定义 2 公共路径，为 FP-tree 节点所共享的树路径。如图 1 中，〈R -> f -> c -> a〉是节点 m、b 的一条公共路径。

定义 3 子条件模式基。频繁一项的条件模式基可由若干部分构成，每一部分对应当前频繁一项的父节点（包括）到树根节点（不包括）所在路径上的所有项集。将每一部分称作该频繁一项的一条子条件模式基，支持度为当前节点的支持度计数。如图 1 所示，m 的子条件模式基有 2 条，分别对应 m 的父节点到根节点 R 的两条路径。

根据上述定义，NHTFPG 搜索条件模式基的过程可描述为：

```
procedure Alg_GetConPatBases( Tree, &Items, conPatBases[ ],
    &baseItems)
```

输入 Tree，频繁模式树。Items，频繁一项集的集合，按支持度计数有序。baseItems，记录公共路径，同时也是当前遍历的树节点项的子条件模式基。

输出 conPatBases，条件模式基的集合。

算法过程描述：

```
{
1)  pChild = root ->firstChild;          //获取 Tree 的第一个孩子
2)    while( pChild ! = NULL) {          //遍历 Tree 的所有孩子
3)    step = 0;                          //向上回退步数
4)    ptr = pChild ->treeNode;
                                  //孩子指针所指的 FP-tree 节点
      //若 ptr 是单支节点则一直向下遍历
5)    while( ptr is Single branch node) {
6)      item = ptr ->item;                          //获取项值
7)      support = ptr ->support;     //子条件模式基的支持度
      //找到 item 在频繁一项集 Items 中的下标
8)      indexOfItem = IndexOf( Items, item);
9)          PushBack ( conPatBases [ indexOfItem ], baseItems,
            support);
      //存储 item 的子条件模式基
10)       Push( baseItems, item);      //追加 item 到公共路径尾
11)     step ++;                                  //回退步数加 1
12)     ptr = ptr ->treeNode;              //向下遍历单支节点
13)   }
14)  execute step 6) - 10);
      //到达叶子节点或新的分叉节点递归求解子条件模式基
15)  GetConPatBases( ptr, Items, conPatBases, baseItems);
16)  Pop( baseItems, step);              //公共路径回退 step 个值
17)  pChild = Tree ->nextChild;     //求解 Tree 的下一个孩子
18) }}
```

以图 1 为例，说明递归搜索求取条件模式基的具体过程。开始时公共路径 baseItems 为空，向下遍历到第一个树节点 f，为当前节点。将 baseItems 的值（空）作为 f 的一条子条件模式基，同时追加 f 到 baseItems。接着向下遍历，当前节点变为 c，将 baseItems 的内容（f）作为 c 的子条件模式基。这里得到 f 为 c 的一条子条件模式基，支持度为 3。同时更新 baseItems 为 fc。同理，向下遍历到下一个树节点 a，得到 a 的一个子条件模式基 fc:3。依次向下，当遍历到节点 p 时，baseItems 为 fcam，得 p 的一条子条件模式基 fcam:2，同时更新 baseItems 为 fcamp。因为此时 p 为叶子节点，所以需向上回退 2 步到最近的一个分叉节点 a，同时 baseItems 也需回退 2 个值，其值由 fcamp 变为 fca。接着遍历 a 的下一个孩子 b，得 b 的一个条件模式基 fca:1。依此进行，直到 Tree 的所有孩子节点均被遍历完为止。遍历完成后，conPatBases 中存储了所有频繁一项及其对应的子条件模式基，同表 1 所示。

由上述子条件模式基的搜索过程可知，NHTFPG 并不需要项头表和同项指针，仅需要存储公共路径的辅助变量 baseItems，其最大长度为频繁一项集的长度。所有频繁一项集的子条件模式基求取完成后，分别对其建立子条件模式树，进行递归挖掘，可得到全部频繁模式。

2 算法性能分析及验证

为比较改进后算法的性能，从时间性能和空间性能两个方面对 NHTFPG 和 FP-Growth 算法做对比分析。算法在 HP Unix 系统下用标准C ++实现，服务器配置为 40 核 CPU，单核主频为 1.1 GHz，内存为 80 GB。所采用的测试数据集为 T10I4D100K。该数据集共 10 万条事务，870 个不同的项，最大事务长度为 29，平均事务长度为 10。

2.1 算法时间性能分析及验证

FP-Growth 算法主要的时间开销是搜索条件模式基，其时间复杂度为 $O(N^2)$，其中 N 为频繁模式树节点数。而 NHTFPG 采用自顶向下的方式遍历频繁模式树，当树遍历结束时即可得到全部频繁一项集的条件模式基，其时间复杂度为 $O(N)$。

从不同的支持度和最大频繁项集阶数两方面对 NHTFPG 和 FP-Growth 的时间性能进行对比。此外,为减少因主机 CPU 负荷等客观因素对算法执行时间的影响,对同一个参数分别运行算法 5 次,去掉最大值和最小值,取其余 3 次时间的平均值作为算法运行时间。结果如图 2 ~3 所示。

由图 2 可知,NHTFPG 在支持度高于 0.5‰时相对于 FP-Growth 的优势并不明显,两者运行时间的差值基本保持平衡。随着支持度降低,FP-Growth 挖掘频繁项集的时间急剧增加,而 NHTFPG 的变化相对平缓。当支持度降为 0.1‰时,NHTFPG 和 FP-Growth 所需时间分别为 61.13 s 及 1 160.7 s,即 NHTFPG 约只需 FP-Growth 所需时间的 1/20,且随着支持度降低,两者的区别越明显。

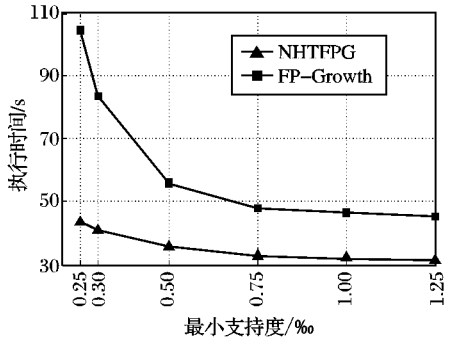

图 2　算法在不同支持度下时间性能对比

图 2 中所选取的 6 档不同的支持度下所得到的最大频繁项集阶数均为 10,为对比 NHTFPG 和 FP-Growth 在挖掘不同最大频繁项集阶数下的时间差异,选取了 6 档不同的最大频繁项集阶数进行对比,对比结果如图 3。

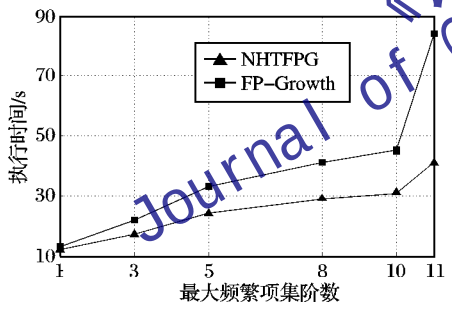

图 3　算法在挖掘不同阶数频繁项集下时间性能对比

由图 3 可知,随着挖掘阶数的增加,FP-Growth 算法耗时呈非线性的急剧增长趋势,而 NHTFPG 耗时变化则相对平缓,这主要是因为 NHTFPG 所采用的自顶向下递归遍历频繁模式树的方式,减少了对公共路径的多次重复遍历,使得挖掘时间大大缩短了。在对通信网络告警数据库的挖掘中发现,针对无线网元告警密集,生成的事务压缩程度高,项集长度长等特点,利用 NHTFPG 算法可在较短的时间内得出频繁项集结果。

2.2　算法空间性能分析

不同于 FP-Growth 算法依据项头表自底向上和自左向右地求取条件模式基的遍历方式,NHTFPG 因采用自顶向下递归遍历(条件)频繁模式树的方式求取条件模式基,从而节省了存储项头表和指向父节点及同项节点指针的开销,使得存储空间开销优于 FP-Growth 算法。

设频繁一项集数为 M,树节点数为 N,项值和指针空间各占 B 个字节,则 NHTFPG 在单层递归中总共节省的存储空间 S = 项头表空间 + 父节点和同项指针空间 $= 2B \times M + 2B \times N = 2B(M+N)$。一般 $N \gg M$,所以 $S \approx 2BN$。可见 NHTFPG 节省的存储空间随树节点的增/减而线性增/减。另外,在 NHTFPG 的每一层递归调用中,条件模式树动态生成和释放,即当前层条件模式树的释放并不影响下一层调用时对条件模式基的求取。在递归挖掘的任意时刻,至多保留了一棵频繁模式树。

3　在通信告警关联规则挖掘中的应用

通信网络每天会产生大量的告警数据,海量的告警数据背后蕴含了大量反映通信网络运行状况和规律的有用信息。利用数据挖掘的方法,可发掘出其中隐藏的告警关联规则,进而指导网络管理人员提高通信网络质量。

将原始的告警数据通过预处理和加窗后转化为事务型的告警事务数据库,然后利用 NHTFPG 算法挖掘出频繁出现的模式,进而导出关联规则。对一个频繁 k 项集将可以导出 $2^k - 2$ 条规则,这样直接导出的规则质量不高,不利于验证规则的正确性和有效性。本文的思路是,先去除交叉支持模式后再导出满足最小置信度的精简的告警关联规则,从而使导出的告警关联规则具有很强的相关性。

定义 4　交叉支持模式。

若频繁模式 $X = \{x_1, x_2, x_3, \cdots, x_k\}$ 的支持度比率 $r(X) = \dfrac{\min(s(x_1), s(x_2), \cdots, s(x_k))}{\max(s(x_1), s(x_2), \cdots, s(x_k))}$ 小于预定义的最小阈值,则称 X 为交叉支持模式。其中,$s(x_i)$ 为项 x_i 的支持度。

交叉支持模式在具有倾斜支持度分布[10]的数据集中频繁存在,从而使导出的规则呈现弱相关性。以网元 JNB1 为例,通过对告警数据生成的告警事务数据库统计,发现项的支持度分布趋势大致如图 4 所示,符合具有倾斜支持度分布数据集的特征。可通过设置一个合理阈值来有效消除交叉支持模式,使导出的规则呈现强相关性。

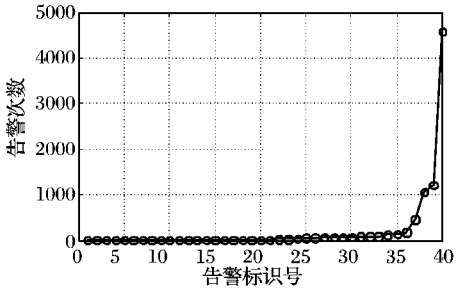

图 4　网元 JNB1 告警计数分布趋势

定义 5　置信度。规则 $\{x_1\} \Rightarrow \{x_2, x_3, \cdots, x_k\}$ 的置信度定义为:

$$confidence\{\{x_1\} \Rightarrow \{x_2, x_3, \cdots, x_k\}\} = \frac{s(X)}{s(x_1)}$$

性质 1　置信度具有反单调性。设规则 $\{x_1\} \Rightarrow \{x_2, x_3, \cdots, x_k\}$ 的置信度为 $conf_1$,规则 $\{x_1, x_2\} \Rightarrow \{x_3, \cdots, x_k\}$ 的置信度为 $conf_2$,则 $conf_1 \leqslant conf_2$。

在实际的关联规则挖掘中,首先去除频繁模式中的交叉支持模式,然后对每一个频繁模式导出所有 1 ->n 形式的满足最小置信度要求的告警关联规则。根据性质 1,由 1 ->n 形式的规则可直接得到 2 ->(n - 1) 形式的规则,且同时满足最小置信度要求。即 1 ->n 形式的规则是所有满足最小置信

度的规则的精简表示形式。

经过上面的分析和处理,选择爱立信的四类网元设备的告警进行关联规则挖掘,挖掘结果如表 2 所示。交叉支持度阈值为 0.01,最小置信度为 0.8。

目前已确认的爱立信告警关联规则共 36 条,挖掘导出的关联规则覆盖了其中 30 条,覆盖率达 83.3%,达到了较好的挖掘效果。另外,根据已有的告警数据库,挖掘出了大量有价值的潜在告警关联规则。

表 2 爱立信关联规则挖掘结果

网元类别	类别名称	告警数	导出规则数
STP	信令转接点	165 989	107
MSC-Server	移动交换中心	127 066	285
TMSC	汇接局移动交换中心	40 249	40
MWG	媒体网关	23 235	62

4 结语

本文提出并实现了基于递归回溯的无项头表的频繁模式挖掘算法 NHTFPG,解决了 FP-Growth 算法对频繁模式树的公共路径造成重复遍历的问题。理论分析和实验结果表明 NHTFPG 在时间性能和空间性能上均优于 FP-Growth 算法。在通信网络告警关联规则挖掘中的应用表明,NHTFPG 可快速挖掘出符合要求的告警关联规则。尽管加上交叉支持模式约束和置信度约束,使得生成的关联规则大大精简,但仍存在较多的冗余甚至无效的规则。因此,深入分析通信网络及其告警的特征,将挖掘算法与专业领域的知识相结合,使得生成的规则更精简,更有效,还有很多工作要做。此外,将 NHTFPG 生成的有效告警关联规则和专家系统相结合,以实现有效的告警处理和快速故障精确定位是进一步研究的方向。

参考文献:

[1] HÄTÖNEN K, KLEMETTINEN M, MANNILA H, *et al*. TASA: Telecommunications alarm sequence analyzer or how to enjoy faults in your network [C]// IEEE/IFIP 1996 Network Operations and Management Symposium (NOMS'96). Washington, DC: IEEE, 1996: 520 – 529.

[2] 肖海林. 网络告警关联规则挖掘系统的研究与设计[D]. 成都: 电子科技大学, 2007.

[3] 朱扬勇, 熊赟. DNA 序列数据挖掘技术[J]. 软件学报, 2007, 18(11): 2766 – 2781.

[4] AGRAWAL R C, AGRAWAL C C, PRASAD V V V. Depth first generation of long patterns [C]// Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM press, 2000: 108 – 118.

[5] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining association rules between sets of items in large databases [C]// Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. Washington, DC: ACM press, 1993: 207 – 216.

[6] HAN J W, PEI J, YIN Y W. *et al*. Mining frequent patterns without candidate generation: A frequent-pattern tree approach [J]. Data Mining and Knowledge Discovery, 2004, 8(1): 53 – 87.

[7] CHRISTIAN B. An implementation of the FP - growth algorithm [C]// Proceedings of the 1st International Workshop on Open Source Data Mining. New York: ACM Press, 2005: 1 – 5.

[8] LIU G M, LU H J, YU J X, *et al*. AFOPT: An efficient implementation of pattern growth approach [EB/OL]. [2010 – 08 – 30]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.1757&rep=rep1&type=pdf.

[9] 周钦亮, 李玉忱, 公爱国. 一种新的高效生成 FpTree 条件模式基的算法[J]. 计算机应用, 2006, 26(6): 1118 – 1121.

[10] 范明, 范宏建. 数据挖掘导论[M]. 北京: 人民邮电出版社, 2006: 237 – 240.

(上接第 1381 页)

4 结语

针对目前已有方法存在着流程挖掘的基本结构有限、抗噪能力弱、计算耗时长等问题,本文提出了一种基于相邻事件概率统计的流程挖掘方法。首先阐述了挖掘规则,然后描述了算法的思想和方法的实现步骤,最后通过流程结构正确性、抗噪和挖掘耗时等性能指标,与 α 算法、启发式算法进行了对比分析,并在 ProM 框架下进行了实验验证。结果表明,该方法不仅可以挖掘出顺序、选择、并行、循环短循环和递归结构等流程结构,而且可以减少噪声对挖掘正确流程的影响度,计算复杂度低。当然本方法还有不足之处,对复杂流程结构或噪声含量≥20%时,本方法的日志流程挖掘能力受限,这将是今后研究的重点。

参考文献:

[1] 赵卫东, 范力. 工作流挖掘研究的现状与发展[J]. 计算机集成制造系统, 2008, 11(12): 2289 – 2296.

[2] RAKESH A, DIMITRIOS G, FRANK L. Mining process models from workflow logs [C]// Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology. Berlin: Springer-Verlag, 1998: 469 – 483.

[3] van der AALST W M P, AJMM W, MARUSTER L. Workflow mining: Discovering process models from event logs [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(9): 1128 – 1142.

[4] AJMM W, van der AALST W M P. Process mining: Discovering workflow models from event-based data [C]// BNAIC2001: Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence. Maastricht: BNVKI, 2001: 283 – 290.

[5] AJMM W, van der AALST W M P. Rediscovering workflow models from event-based data using little thumb[J]. Integrated Computer-Aided Engineering, 2003, 10(2): 151 – 162.

[6] COOK J E, WOLF A L. Discovering models of software process from event-based data[J]. ACM Transactions on Software Engineering and Methodology, 1998, 7(3): 215 – 249.

[7] van DONGEN B, de MEDEIROS A, VERBEEK H, *et al*. The prom framework: A new era in process mining tool support [C]// Proceedings of Application and Theory of Petri Nets 2005. Berlin: Springer-Verlag, 2005: 444 – 454.

[8] ROZINAT A, van der AALST W M P. Conformance testing: Measuring the alignment between event logs and process models [EB/OL]. [2010 – 05 – 10]. http://cms.ieis.tue.nl/Beta/Files/WorkingPapers/Beta_wp144.pdf.

[9] ALVES A K, de MEDEIROS A, UNTHER C W G. Process mining: Using CPN tools to create test logs for mining algorithms [EB/OL]. [2009 – 08 – 11]. http://is.tm.tue.nl/researeh/processmining/tools/ProM/cpnToolConverter.zip.