一种基于立方体的复杂查询的高效算法*

曾德胜[1]，覃　泽[2]，王日凤[3]，张师超[3,4]

(1. 罗定职业技术学院　电子信息系，广东 罗定 527200；2. 南宁地区教育学院，广西 南宁 530001；3. 广西师范大学 计算机系，广西 桂林 541004；4. 悉尼科技大学 信息技术学院，澳大利亚)

摘　要：提出一种高效的整体多特征方查询算法。该算法首先将数据立方体水平分块成多个小数据集，然后将各子查询中的聚集函数分类，并对其中的分布和代数聚集函数使用分布聚集特性优化计算，使得整体多特征方查询可以局部使用分布多特征方查询的优化计算方法。实验结果证明该方法可以有效地提高整体多特征方查询的效率。

关键词：复杂查询；多特征方；多粒度聚集

中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2007)03-0030-04

Fast Algorithm for Complex Queries Based on Data Cube

ZENG De-sheng[1], QIN Ze[2], WANG Ri-feng[3], ZHANG Shi-chao[3,4]

(1. Dept. of Electronic Information, Luoding Vocatronal Technology College, Luoding Guangdong 527200, China; 2. Nanning Education College, Nanning Guangxi 530001, China; 3. Dept. of Computer Science, Guangxi Normal University, Guilin Guangxi 541004, China; 4. Faculty of Information Technology, University of Technology Sydney, Australia)

Abstract: This Paper proposed a fast algorithm for Holistic MF-Cubes queries, which included partition datacube into smaller sub-cubes and classify aggregating functions and identify the distribute ones, which can use the coarse-finer granularities computing property. Experiment shows that this algorithm is highly efficient to the Holistic MF-Cubes query.

Key words: complex queries; multi-feature cube; multi-granulated aggregation

0　引言

近年来，对多维空间(Multi-Dimensional Space, MDS)上的数据分析成为研究的热点。多维数据分析有着广泛的应用前景，如可以实现趋势预测、发现异常模式，以及回答 What-If 问题等。由于多维数据表的数据特点和数据复杂性，需要分析多个属性间的关系以及对多个维属性同时进行操作。多维数据模型采用数据立方体(Data Cube)形式。数据立方体表示多维数据的优势在于它能在多个粒度层上表示数据，同时有利于计算聚集值。基于数据立方体的查询可以方便地进行各种 OLAP 操作[1]，如上卷、下钻、切片、切块等。由于海量数据所带来的问题，完全物化整个数据立方体是不现实的。目前很多研究工作均集中在如何有效减小立方体大小，提高聚集计算速度，从而快速响应查询等方面。这些研究一般针对的都是简单查询。简单查询大多可以直接使用分布聚集函数(如 sum()，count()等)或代数聚集函数(average()等)实现。随着数据挖掘技术的发展和商业竞争的需求，多粒度的复杂聚集研究已经成为数据立方体技术进一步发展的方向[2]。多粒度的复杂聚集是基于立方体的复杂查询的基本特征。相对于简单查询，复杂查询能提供更多和更详细的数据信息，对决策分析提供强有力的支持。现有的决策支持系统追求的目标就在于如何有效地回答大型数据库的复杂查询。立方体查询之所以不同于一般的 SQL 查询，就在于它允许在多个粒度上计算聚集值。复杂查询实例如下：

查询 1　按{month, customer, item}的所有分组，求出 2004 年的最低价格，并在最低价格的元组中，找出销售量最大和最小的商品利润总和。

查询 2　按{month, customer, item}的所有分组，求出 2004 年所有分组的最低价格，并求出各分组中商品价格在最低价格的 125%、150%、175% 以内的商品总销售量。

上述立方体查询将会在八个不同粒度上计算聚集值[3]，分别是{month}、{customer}、{item}、{month, customer}、{month, item}、{customer, item}、{month, customer, item}和{*}。其中，“*”表示所有(ALL)。这里的每个粒度，就对应着 SQL 中的一个 Group By 分组。

根据上述复杂查询的实例，可以归结出基于立方体的复杂查询具有这样一些特点：①查询在所有可能的 2^n(n 为分组维属性的个数)个粒度上进行计算；②复杂查询由多个具有聚集依赖关系的简单查询构成，这些简单查询构成了复杂查询的子查询。文献[4]将这种在所有可能的 2^n 个粒度进行的具有聚

收稿日期：2005-12-22；修返日期：2006-06-25　　基金项目：澳大利亚 ARC 项目(DP0559536, DP0667060)；国家自然科学基金资助项目(60463003)

作者简介：曾德胜(1980-)，男，广西武宣人，硕士，主要研究方向为数据挖掘；覃泽(1958-)，男，广西玉林人，副院长，副教授，学士，主要研究方向为计算机应用；王日凤(1974-)，女，广西柳州人，硕士研究生，主要研究方向为数据挖掘、数据库；张师超(1962-)，男，广西桂林人，教授，博导，博士，主要研究方向为数据挖掘、人工智能.

集依赖关系的复杂查询称为多特征立方体(Multi-Feature Cubes, MF-Cubes)查询。多特征方查询依据查询中所使用的聚集函数类型进行分类。在文献[1]中将聚集函数类型分为三类,即分布(Distributive)、代数(Algebraic)和整体(Holistic)。相应地,多特征方也分为分布多特征方、代数多特征方和整体多特征方三类。

分布多特征方查询有一个明显特征,那就是粗粒度的聚集结果可以由细粒度的聚集结果计算得到,如查询 1 为分布多特征方查询的典型实例。假设在细粒度{month, customer, item}上计算了等于最低价格的各个月份的销售量总和,则粗粒度{customer, item}的销售量总和就等于细粒度{month, customer, item}中计算的 12 个月的销售量相加,不必重新计算。本文称这个特征为分布聚集特性,用于描述两个粗细粒度间的聚集计算依赖关系。代数多特征方通过变化聚集函数也可转换为分布多特征方,也可以使用这种聚集计算特性来实现计算的优化,如 count()函数可以由 sum()/n 计算得出。

整体多特征方就不能直接使用分布聚集特性来优化计算。例如,查询 2 为整体多特征方查询的实例。细粒度{month, customer, item}的查询结果不能用于计算粗粒度{customer, item}的查询结果。已经计算了粒度{month, customer, item}的最小价格 MIN(price)及各个子查询的销售总和 SUM(R1.sales), SUM(R2.sales), SUM(R3.sales)。假设计算出 1 月份的最低价格是 120 美元,在最低价格 150% 以内(120～180 美元)的商品的销售总和为 2 000 美元,而整年的最低价格是 110 美元,则在最低价格 150%内的价格取值区间是110～165 美元。无法简单地将 12 个月的销售总和相加得到整年的销售总和,因为无法知道 1 月份的 2 000 美元中哪些是在 110～165 美元。

对于分布和代数多特征方查询,由于可以直接使用分布聚集特性来简化计算,对此已经提出了许多有效的实现技术,如 Partitioned-Cube 算法[5]、BUC 算法[6]以及 OVERLAP 算法[3]等。因为整体多特征方需要对各个子方独立计算各粒度的聚集值,计算复杂度和计算代价较前两者高出许多,所以,需要为整体多特征方查询寻找更有效的实现方法。文献[4]中也简单地提到可以用 Partitioned-Cube[5]算法中的数据分块方法来实现,该算法是实现分布和代数立方体查询的经典算法。Partitioned-Cube 算法采用水平分块方法将数据立方体分块为多个适合内存大小的子方,然后在每个子方递归地实现分布和代数立方体查询聚集计算。对于整体多特征方,需要对分块得到的子方独立计算各粒度上的聚集值,分别回答各个粒度上的子查询。这种独立计算各个粒度上聚集值的代价无疑是巨大的,特别是当分组属性较多的时候。有没有更有效的解决方法呢?

考虑到复杂查询是由具有聚集依赖关系的多个简单查询构成,因此,复杂查询中的聚集函数也可由简单查询的聚集函数组成,一般为分布聚集函数、代数聚集函数和整体聚集函数。分布和代数多特征方中不会含有整体聚集函数,而整体多特征方则很有可能是它们的组合,如查询 2 中的聚集函数就是分布聚集函数 MIN()和用户自定义函数,如 1.75MIN()。根据这个特点,可以先将子查询的聚集函数分类,含分布和代数函数的子查询可以使用分布聚集特性来优化计算,以提高复杂查询的性能。同时,若自定义函数中也含有分布函数,则也可以进行聚集优化计算。

例如,查询 2 中共有子四个查询,其中第一个子查询为求最小价格,这是一个分布聚集函数,因此在计算每个子方时,多个粒度间可以使用分布聚集特性实现,即粒度{AB}的最小价格可以由粒度{ABC}的各组最小值直接计算得到,而不必重新计算。其后的三个查询是分布函数和代数函数之外的自定义函数,也并非需要如文献[5]所说的那样完全独立计算每个粒度的聚集值。虽然不能将粒度{month, customer, item}的计算结果直接用于计算,但是假设粒度{month, customer, item}中有 6 个月的最小价格 MIN(price)与粒度{customer, item}的最小价格是一致的,则在求粒度{customer, item}的计算过程中,就不需要重新计算这相同的 6 个月的聚集值,只需要计算余下 6 个月的聚集值即可。本文称这两种优化计算过程为局部聚集优化特性。

本文正是基于局部优化特性对整体多特征方进行查询计算优化的。这是首次提出的对整体多特征方的优化算法。优化后的算法与 Partitioned-Cube 算法在模拟数据集上进行对比实验,优化算法 ImpPartitioned-Cube 查询效率的提高可达到 90%以上。

1　整体多特征方查询实现算法及其描述

算法的第一步是将数据立方体进行分块处理,即将立方体分成多个子方,在各个子方上计算粒度聚集值。分块是由于数据集太大,内存无法一次处理完而采取的基本方法。对于立方体分块,一般有两种分块方法:①垂直分块方法,如 Array-based Cube 算法[7],即按立方体中所有分组维属性的值均匀分块。这样的分块方法不能使同一个分组的数据在同一个块,不适合于粒度的分开计算。②水平分块方法,即 Partitioned-Cube[5]的分块方法,按某个分组维属性的值分块,使得同一个分组的所有数据同在一个块中。例如,分组维属性有月份和地区,可以先按月份分块,最多分 12 个块,每个块为一个月份的数据,若每个月份数据量太大,则对每个月份的数据再按地区分块。这种分块方法确保了同一个分组的数据在同一个块中,因此适合于粒度的独立计算。

对于整体多特征方查询的实现,一个直接的办法就是采用文献[5]的 Partitioned-Cube 算法,水平分块后独立计算每个粒度上的聚集值。由于 Partitioned-Cube 完整算法是为实现分布和代数这两种立方体查询的,先将其修改为适合整体多特征方计算的 ModiPartitioned-Cube 算法。

1.1　ModiPartitioned-Cube 算法实现整体多特征方查询

ModiPartitioned-Cube 算法思想主要有以下两点:①将数据立方体分化为多个适合内存大小的子方(Memory-Cube);②在每个子方上独立执行各个复杂的查询操作(Independent Operation)。修改后的算法描述如下:

Algorithm ModiPartitioned-Cube(R, {B_1, …, B_m}, A, G)
INPUT R: 基本方体;
　{B_1, …, B_m}: 立方体的分组维属性 B_1, …, B_m
　A: 用于聚集的测试属性
　G: 聚集函数
OUTPUT:
Result of Complex Queries: 复杂查询的结果
METHOD:

```
(1) Dim = B_i ∈ {B_1, …, B_k}; n = cardinality(B_i);
(2) R_1, …, R_n = Partition-Cube(R, Dim, n, {B_1, …, B_k});
    //按属性 Dim 分块立方体 R,子方个数 n
(3) for(i = 0; i < n; i++)
//n 个子方,逐个计算粒度上的聚集,依据查询中的聚集函数 G 进行
(4) {
(5) initialize accumulators for aggregating at granularities which include B_i; //初始化含 B_i 的聚集器
(6) SortR = Sort(R_i, B_i) //按粒度维属性值排序,从最细粒度开始
(7) for all queries
(8) accumulators = GranularityAggregate (SortR, GF(A))
    //计算各子查询
(9) }
(10) R_1, …, R_n = Partitioned-Cube(R, Dim, n, Countdata[Dim], {B_1, …, B_{i-1}, B_{i+1}, B_k});
(11) repeat (3) to (9);
    //将含 K-1 个分组属性的基本表重复前面的操作过程
```

算法中步骤(1)选择一个分块维属性 B_i;(2)为一个分块程序实现将 R 按 B_i 分块;(3)～(9)是计算含分块维属性的粒度分开计算;(10)、(11)是对不含分组维属性 B_i 的粒度分开计算,只需要将去除 B_i 属性后的立方体重复前面的过程即可,对没有计算的粒度的实现过程也同步骤(10)和(11)。算法略。

如前所述,ModiPartitioned-Cube 算法的实现方法需要独立计算各个粒度上的聚集值,当立方体很大时,计算复杂,时间开销大,查询效率低。特别是当分组属性很多时,实现这类复杂查询的代价会更大。本文采用优化策略正好可以有效弥补这种不足。

1.2 改进算法 ImpPartitioned-Cube 算法思想及描述

1.2.1 算法思想

ImpPartitioned-Cube 算法思想主要有:①将数据立方体分块为多个适合内存的子方;②对各个子方,使用局部聚集优化特性计算多个粒度(Dependent Operation)。

下面分别叙述算法处理过程:

(1) 立方体分块处理

算法的第一步与 ModiPartitioned-Cube 算法一致。首先,按分组维属性 B_i($B_i \in \{B_1, \cdots, B_m\}$),将立方体水平分块为 n 个子方 $R_1, \cdots, R_n$,这 n 个子方含全部分组维属性 $B_1, \cdots, B_k$;若 cardinality(B_i)足够大,则得到的子方所含元组数为 T/n。其中 T 为立方体的元组总数,n 不大于 B_i 的基数,即 $n \leq$ cardinality(B_i)。例如,若按分组属性 A 分化,假设 cardinality(A) = 20,则基本表第一次分化的子方个数不超过 20。对每个按分组维属性分化得到的子方 R_i,检测其大小,若仍然很大,则进行二次分化,继续选择下一个分组维属性,分化的方法一样。

对分化得到的每个子方,分别进行各粒度上的查询聚集计算。按分组维属性分化得到的所有粒度分成两组:

一组含分组属性 B_i,可以在这 n 个子方上计算出来。聚集计算之前,先按粒度中的维属性进行排序。对于排序,本文采用了先计算细粒度再计算粗粒度的计算次序,这样做可采用 PIPESORT[3] 算法中的共享前缀排序技术。例如,若先对细粒度{ABC}进行排序和操作,然后计算粒度{AB},就无须再对 AB 排序,直接可以共享前面排序的结果。对于不含分组维属性 B_i 的各粒度计算,可以有两种方法:①重新遍历每个子方,分别计算各个子方数据;②将 R 再次分化,重新读取数据。此次分化的 R 含($K-1$)个分组维属性(不含 B_i),实现方法与前面含 K 个分组属性的过程一样。我们采用的是方法②,因为它分化得到的子方要少于方法①,这样可以减少遍历的子方数目。

(2) 按子查询聚集函数分类处理对分布和代数聚集函数使用局部聚集优化特性优化聚集计算

在确定粒度计算顺序后,为了使用分布聚集特性,将各个子查询中的聚集函数进行分类:一类为分布或代数函数;另一类为整体聚集函数。先计算前者,这样可以利用使用部分聚集特性来优化计算,再计算整体聚集值。例如查询 2,计算最小价格时就可以使用分布聚集特性,计算后三个子查询时采用的是逐个测试的方式,以确认能否使用局部的分布聚集特性。

例如,分组维属性为{A, B, C},其粒度个数为八个,执行这八个粒度上的聚集计算需要以下过程:①先按 A 分块,则得到的子方可逐次计算含 A 的三个粒度 ABC、AB 和 A。②对于含属性 B 的粒度计算,则是将数据集去掉属性列 A 后,对余下的数据集进行操作。若仍大于内存,则按属性 B 进行分块,对分块后的子方逐次计算粒度 BC 和 B。③若在②中不需要分块,则直接对其中的立方体计算粒度 C,采用类似②的操作,去掉属性列 A 和 B,仅含分组维属性 C。若还大于内存,则按 C 分块,计算粒度 C。④若在③中不需要对 C 分块,则直接对其中的数据集计算粒度 ALL;否则去掉所有的分组维属性 A、B、C,对余下的数据集进行类似②的操作,计算粒度 ALL。

1.2.2 ImpPartitioned-Cube 算法描述

```
Algorithm ImpPartitioned-Cube(R, {B_1, …, B_m}, A, GFs)
INPUT  : 基本立方体;
         {B_1, …, B_m}: 立方体的分组维属性 B_1, …, B_m
         A: 用于聚集的测试属性
         GFs: 聚集函数集
OUTPUT:
     Result of Complex Queries: 复杂查询的结果集
METHOD:
(1) Dim = B_i ∈ {B_1, …, B_k}; n = cardinality(B_i);
(2) R_1, …, R_n = Partitioned-Cube(R, Dim, n, {B_1, …, B_k});
    //按维属性 Dim 分块立方体 R、子方个数 n
(3) for(i = 0; i < n; i++) //对 n 个子方执行如下操作
(4) {
(5) initialize accumulators for aggregating at granularities which include B_i; //初始化含 B_i 的聚集器
(6) SortR = Sort(R_i, B_i) //对粒度 B_i 按粒度维属性排序
(7) for distribution queries
(8) accumulators = PartDistribAggregate(SortR, GF(A));
    //先计算分布和代数聚集函数的子查询
(9) for rest sub-queries
(10) accumulators = GranularityAggregate (SortR, GF(A))
    //后计算余下的子查询
(11) } //for
(12) R_1, …, R_n = Partitioned-Cube(R, Dim, n, {B_1, …, B_{i-1}, B_{i+1}, B_k});
(13) repeat (3) to (11);
    //将含 K-1 个分组属性的基本表重复前面的操作过程
```

算法中步骤(2)实现将 R 分块,按 B_i 分化,若基本立方体无须分块,则分块个数 $n=1$;(3)～(11)是对分块后的子方计算含维属性 B_i 的所有粒度;(6)先对子方排序,为使用共享前缀排序技术,先按最细粒度的维属性排序;(7)和(8)是对含分布和代数聚集函数的子查询使用局部聚集优化特性计算;(9)和(10)是对余下子查询计算聚集值;(12)和(13)对不含分组维属性 B_i 的粒度进行计算,只需要将去除 B_i 属性后的立方体重复前面的过程即可,对余下没有计算的粒度,也是重复(12)

和(13)。算法略。

2　实验和结果分析

算法用VC ++ 6.0 实现,采用随机函数生成四组均匀分布的模拟数据集,元组数为 100 万 ~500 万。数据表含五个维属性,分组维属性个数为 3,详细参数如表 1 所示。在内存为 2.0 GHz、CPU 主频为 2.6 GHz、操作系统为 Windows 2000 的 Dell Workstation PWS650 计算机上进行实验,将 ImpPartitioned-Cube 算法与 Partitioned-Cube 算法进行比较。假设将大型数据集分块一次得到的子方均适合内存。实验的目的主要是比较 ImpPartitioned-Cube 算法与 ModiPartitioned-Cube 算法的性能,两种算法分块的次数与子方个数是相同的,排序过程也是相同的,故实验结果中的时间不含立方体分块时间和排序时间,是复杂查询实现各个子查询的聚集计算的时间效率。用两种算法对复杂查询 2 的查询效率进行比较,实验结果如图 1 所示。

表 1　实验数据参数

属性	分组维属性			维属性		测试属性
	A	*B*	*C*	*D*	*E*	*F*
Cardinality(基数)	10	100	100	100	100	10^4

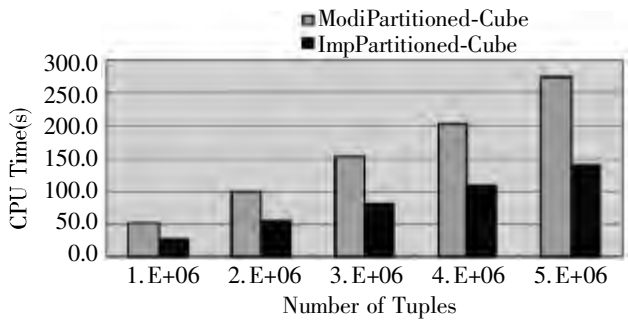

图 1　ImpPartitioned-Cube 与 ModiPartitioned-Cube 算法的性能比较

从图 1 中可以看出,使用了局部分布聚集特性计算的算法性能提高很明显,最低也提高了 80%,高时达到 90% 以上。提高的主要原因在于采用了局部聚集优化特性优化计算过程后,简化了聚集计算过程,使遍历数据量大大减少。实验实现复杂查询 2,有一个分布函数,遍历数据量只有 ModiPartitioned-Cube 算法的 45% 左右。

3　结束语

本优化算法是针对整体多特征方的优化算法,其优越性主要有:①将立方体进行水平分块,适合粒度的分开计算;②将各子查询的聚集函数分类,对分布函数使用分布聚集特性进行查询的优化,改进了原来的整体多特征方不能用分布聚集特性的不足。在模拟数据集上进行实验验证,结果证明使用局部分布聚集特性优化计算后的整体多特征方算法的性能显著提高。

参考文献:

[1] RAY J, BOSWORTH A, LAYMAN A, *et al*. Datacube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals: proc. of the IEEE ICDE[C]. [S. l.]: [s. n.], 1996: 152-159.

[2] HAN J, KAMBER M. Data mining: concepts and techniques[M]. [S. l.]: Morgan Kaufmann Publishers, 2001.

[3] AGARWAL S, AGRAWAL R, DESHPANDE P M. On the computation of multidimensional aggregates: proc. of Int'on Conf. Very Large Data Bases[C]. [S. l.]: [s. n.], 1996: 506-521.

[4] ROSS K A, SRIVASTAVA D, CHATZIANTONIOU D. Complex aggregation at multiple granularities: EDBT'98[C]. [S. l.]: Springer-Verlag, 1998: 263-277.

[5] ROSS K, SRIVASTAVA D. Fast computation of sparse datacubes: VLDB'97: proc. of Int. Conf. on Very Large Data Bases[C]. Athens: [s. n.], 1997: 116-125.

[6] BEYER K S, RAMAKRISHNAN R. Bottom-up computation of sparse and iceberg cubes: proc. of ACM SIGMOD Conf. [C]. Philadelphia: [s. n.], 1999: 359-370.

[7] ZHAO Y, DESHPANDE P M, NAUGHTON J F. An array-based algorithm for simultaneous multidimensional aggregates: proc. of ACM SIGMOD Conf. [C]. Tucson: [s. n.], 1997: 159-170.

(上接第 29 页)

表 2　不同模型下的执行时间

*E / *T / *L	10	25	50	100
10	270	332	756	1 624
50	1 384	1 794	3 650	8 230
100	3 176	3 681	7 570	16 890
500	16 240	17 330	37 230	86 490

为了对算法进行全面的评估,实验中 * *T* 最大取值达到 100,能完全满足实际业务流程的需求;* *L* 最大取值达到 500(在这种情况下,日志文件大小约为 10 MB),这样大的样本可以充分测试算法的性能。从表 2 中可以明确地得出,改进后的算法仍具有很好的性能,可以很好地满足大多数具有复杂模式的实际业务流程的重构工作。

5　结束语

在文献[1,5,6]中,通过对工作流日志中的事件记录进行分析,提出一种基于工作流事件的重构算法。算法能够对大部分工作流进行重构处理,但在识别具有复杂模式的工作流时,仍存在着不完整性。本文分析了在实际工作流中易出现的复杂模式,探究了文献[1,5,6]算法不能处理这些复杂模式的原因。通过对该算法进行修正,包括重定义日志完整性和事件排序关系,增加相应的预处理阶段和后置处理阶段等,使该算法能够对这些复杂模式进行有效处理。未来还有大量的工作,包括对重复任务、内置库所以及非自由选择等复杂模式的研究。总之,我们希望通过扩展和完善该算法,来处理更多类型的工作流日志。

参考文献:

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

[2] Workflow Management Coalition. The workflow reference model, TC00-1003[R]. [S. l.]: WfMC.

[3] YAN Yuhong, BEJAN A. Modeling workflow within distributed systems: the 6th International Conference on CSCW in Design[C]. London: [s. n.], 2001.

[4] AALST W M P van der, HOFSTEDE A H M ter. Advanced workflow patterns[C]//ETZION O, SCHEUERMANN P. CoopIS 2000: the 7th International Conference on Cooperative Information Systems, volume 1901 of Lecture Notes in Computer Science[C]. Berlin: Springer-Verlag, 2000: 18-29.

[5] AALST W M P van der, WEIJTERS A J M M, MARUSTER L. Workflow mining: discovering process models from event logs[J]. **IEEE Transactions on Knowledge and Data Engineering**, 2004, **16**(9): 1128-1142.

[6] AGRAWAL R, GUNOPULOS D, LEYMANN F. Mining process models from workflow logs: the 6th International Conference on Extending Database Technology[C]. [S. l.]: [s. n.], 1998: 469-483.