基于层次空间聚类的表语义汇总算法

孙 翀 卢炎生
(华中科技大学计算机学院 武汉 430074)

摘 要 通过数据概化,在多维属性的属性值概念分层上构造少量的具有抽象语义的元组来替换大量具有详细语义的原始元组,从而汇总数据表,这称作表语义汇总。给定原始数据表及其多维属性的属性值的概念分层,表语义汇总的目标是产生规定压缩率且保留尽可能多的语义信息的汇总表。现有算法采用在概化元组集合中寻找最佳概化元组组合的策略将其转换成 Set-Covering 问题来解决,尽管采取了多种优化策略(如预处理、分级处理)来提高效率,但仍存在转换开销大、算法框架复杂且不易扩展到高维属性等缺点。通过定义多维属性层次结构的度量空间将该问题转换为多维层次空间聚类问题并引入 dewey 编码来提高转换效率,提出了基于快速收敛的层次凝聚和基于层次空间分辨率调整的两种聚类算法来高效地建立语义汇总表。经真实数据集上的实验表明,新算法在执行效率和汇总质量上都优于现有方法。

关键词 数据概化,概念分层,语义汇总,层次聚类
中图法分类号 TP311 **文献标识码** A

Clustering-based Algorithms to Semantic Summarizing the Table with Multi-attributes' Hierarchical Structures

SUN Chong LU Yan-sheng
(School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract Table semantic summarization is to create a small size summary table by using a few general tuples to replace all tuples in the raw data table with the help of attributes' concept hierarchies. The aim of summarization is to restrict the size of the summary table to a fixed value with the semantic information remained in it as more as possible. The existing method translates table summarization to a set-covering problem and spends much cost in the problem translation which makes it impractical. We defined the metric space of tuples with multi-attributes' hierarchical structure and translated this problem to a clustering problem in a hierarchial space. We proposed two algorithms. One was hierarchial agglomerative method and the other was based on the idea of adjusting the resolution of the hierarchial space. The experiment on real life dataset shows that our methods are better than the existing one in both running time and summary quality.

Keywords Data generalization, Concept hierarchy, Semantic summarization, Hierarchical clustering

1 引言

随着无线网络的普及,许多移动手持设备,如智能电话(SP)、掌上电脑(Pocket PC)和个人数字助理(PDA)被广泛用于网络信息浏览。与有线网络上的 PC 终端不同,手持设备的显示和计算能力有限,而且无线通讯的费用也远高于有线通讯。因此,在这些手持设备上浏览大数据表是非常困难且不经济的。手持设备浏览图片也存在类似的问题。当手持设备浏览网络上的某张大图时,网络代理服务器根据手持设备的硬件能力动态地将原始大图通过降低图片质量(如图像尺寸、分辨率、颜色信息)的方法产生小图片,发送给手持设备。基于这一思想提出了各种缩减数据表大小的表汇总算法。除数据浏览的用途外,汇总表作为大数据集的数据摘要在数据分析与管理领域也有着重要作用:近似查询和查询优化。通过在汇总表上运行查询,可获得近似的查询结果及估算查询在原始表上的选择度。

表语义汇总技术是最新的汇总数据表的技术,其优点是:汇总后的数据表保留了丰富的语义信息,具有良好的可读性,且表的缩减率可随用户的需求进行定制。它利用表中各属性上的属性值概念分层对原始数据表进行元组概化,用少量的"抽象"元组来表示大量的"详细"元组(即原始元组),从而达到缩减数据表的目的。元组概化过程中会产生语义信息的损失,表语义汇总的目标就是构建符合规定压缩率的汇总表且保留尽可能丰富的语义信息。

属性的属性值概念分层是描述属性值域上分类关系的一种树形层次结构,它反映属性的本体信息。如图 1 所给出的概念分层所示:对于分类型的属性 Location,其属性值可以从"城市"概化为较高的"省";对于数值型属性 Age,可通过人

到稿日期:2011-04-01 返修日期:2011-06-05 本文受国家"十一五"部委预研基金(513150402)资助。
孙 翀(1981-),男,博士生,主要研究方向为嵌入式数据管理、数据挖掘,E-mail:Nicksun217@163.com;卢炎生(1953-),男,教授,博士生导师,主要研究方向为现代数据库系统、软件测试、数据挖掘。

工方式建立区间来构建分类关系，如将 18 岁到 19 岁概化为一个区间[10,19]并用 1 * 表示。在概念分层中，节点的位置决定了语义的"详细"程度和属性值的概化能力：位置越接近根，节点的语义信息越少，而其概化的范围越大，任意节点能概化以其为根的子树内的所有节点。

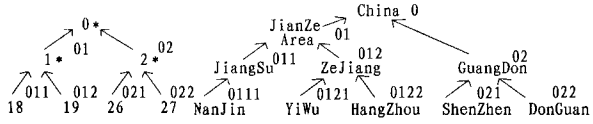

图 1　Age 与 Location 属性的概念分层

以例说明表语义压缩。表 1 是由 Name，Age，Location 3 属性组成的原始数据表(附加列 Tid 用于标识元组)，用于记录 CBA 球员信息，包含 6 条元组。表 2－表 4 为表 1 在 Age 与 Location 属性上进行汇总的缩减率为 3 的语义汇总表(附加列 Gen_Set 标识对应元组所概化的原始元组集，附加列 DeweyCode 为元组的内部编码)，其属性值概念分层如图 1 所示。直观上而言，表 4 蕴含的语义信息最丰富：原始表中球员分布为江浙地区和广东省，江浙的球员较年轻(年纪均小于 20 岁)，而广东球员较成熟(年龄大于 20 岁)。客户若仅对广东的球员感兴趣，可向服务器发送信息，获得表 4 中第二条概化元组对应的所有原始元组；另一方面，表 2 的语义信息最少：所有球员均属于中国。4.2 节将给出这个汇总表的语义定量计算。

表 1　CBA 球员信息表

Name	Age	Location	DeweyCode	Tid
CAI Li-long	18	NanJin	011,0111	t1
JIANG Kai-yi	19	HangZhou	012,0122	t2
ZHANG Da-yu	18	YiWu	011,0121	t3
ZHU Fang-yu	26	ShenZhen	021,021	t4
ZHANG Kai	27	DongGuan	022,022	t5
WANG Shi-peng	26	ShenZhen	021,021	t6

表 2　粗糙的语义压缩表(缩减率 $k=3$)

Name	Age	Location	DeweyCode	Gen_Set
——	*	China	0,0	{t1,t2,t6}
——	*	China	0,0	{t3,t4,t5}

表 3　较精细的语义压缩表(缩减率 $k=3$)

Name	age	Location	DeweyCode	Gen_Set
——	*	China	0,0	{t1,t4,t5,t6}
——	1*	ZheJiang	01,012	{t2,t3}

表 4　最优的语义压缩表(缩减率 $k=3$)

Name	age	Location	DeweyCode	Gen_Set
——	1*	JiangZhe Area	01,01	{t1,t2,t3}
——	2*	GuangDong	02,02	{t4,t5,t6}

为方便描述，下述的表语义汇总称为 K-Summary，其中 K 代表缩减率。值得注意的是，K-Summary 与 K-Anomity 的重要区别在于前者中 K 代表汇总表的压缩率为 K 而无每个概化元组必须概化 K 个原始元组的条件，其关注的是汇总表的总体语义；后者要求每个概化元组必须概化 K 个原始元组，其关注的是数据的隐私。

现有的表语义汇总技术 AlphaSum 将 K-Summary 作为 K-Anonymity[5-7]进行处理，用解决 K-Anonymity 的方法来解决 K-Summary 会导致语义汇总表的质量下降，而且 AlphaSum 存在转换开销大、算法框架复杂且不易扩展到高维属性等缺点。

通过引入 Dewey 编码并定义多维属性层次结构的度量空间，提出利用空间聚类算法来高效地建立语义汇总表的方法。

• 在应用方面，将 K-Summary 建模成空间聚类问题，提出了使用基于多维属性层次结构上的聚类来实现表语义汇总系统。相对于 AlphaSum，新方法具有更好的汇总质量、更快的汇总速度。

• 在算法方面，基于层次凝聚和分辨率调整两种不同的策略来设计出两种聚类算法，优化了传统的层次凝聚算法。相对于传统的"两两凝聚模式"，新的凝聚模式在不影响聚类质量的条件下，每轮可以凝聚更多分簇，具有更快的收敛速度。提出了利用层次空间的分辨率调整策略，并通过调整层次空间的分辨率来聚类分簇。这种聚类算法具有更快的速度，适合大规模数据。

2　相关工作

TabSum[1]是最早的表汇总系统，它通过缩减元组和属性个数对表进行汇总。为了缩减元组个数，它首先依照一个或多个属性做 group 操作，然后将同一个 group 中的多个元组合并成单个元组。合并后的元组中，各属性值存放 group 中属性值的统计信息。为了缩减属性，它采取了值简化(降低值精度)和属性合并(合并多个简单属性)的策略。TabSum 采取表的物理压缩方式对表进行汇总，其缺陷是表的可缩减程度是一定的，压缩后的表可读性较差且损失的信息无法度量。

SaintEtiQ[2,3]最早引入属性值概念分层信息。为了生成抽象质量最优的抽象表，它计算且增量维护原始数据表的各汇总表的集合，并且以概化程度的层次结构整理这些汇总表。在该层次结构中，所有的非叶节点对应其子节点的概化。算法使用该层次结构根据汇总质量自顶向下地寻找原始数据表的最优汇总表。随着汇总表的维数以及概念分层的层数增加，维护 SaintEtiQ 中的汇总表层次结构的空间代价非常大，并且在该结构中寻找最优抽象表的时间代价也非常大。

AlphaSum[4]最早考虑 K-Summary 问题。相对于 SaintEtiQ，其优点是效率上有较大改进，量化了汇总过程的语义损失，表的缩减率可控。

AlphaSum 系统分为预处理与核心处理两阶段，其核心处理阶段如图 2 所示。

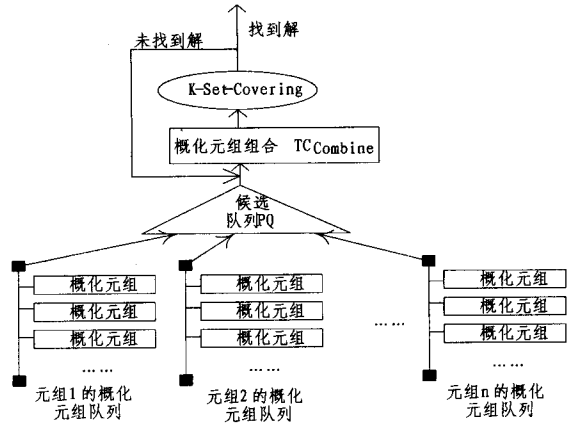

图 2　AlphaSum 的核心处理阶段

预处理阶段：针对原始表中的每个元组，计算出所有能概

化它的概化元组，从而形成原始元组的(以语义损失升序的)概化元组队列，n 个原始元组对应 n 个概化元组队列；构建及维护每个(前步计算出的)概化元组的全局统计信息，该统计信息用于向核心处理步骤提供每个概化元组在原始数据表中所能概化的元组的集合。

核心处理阶段：将 K-Summary 建模成 K-Set-Covering 问题。从 TCcombine 中选择$\lceil n/K \rceil$个概化元组，使得每个概化元组至少能概化 K 个原始元组，且任意两个概化元组对应概化的原始元组集的交集为空。若在当前 TCcombine 中找到满足上述条件的$\lceil n/K \rceil$个概化元组，则算法结束；否则从候选队列 PQ 中选择下一个概化元组，将其添加到当前 TCcombine 并进行下一轮的寻找。而 PQ 也相应地从某个概化元组队列中选择队首项进行补充，直到某个概化元组队列为空，从而结束算法。

如前所述，核心算法优先考虑了语义损失小的概化元组的组合，保证了汇总表的语义质量。

K-Set-Covering[11]是经典的 NP 问题之一，其解空间具有如下性质：就语义质量而言，任意一个$(K,2K)$-Set-Covering的解(即每个概化元组概化 K 到 $2K$ 个原始元组)最多是 K-Set-Covering 最优解的$(1+\ln 2K)$倍。利用该性质，AlphaSum 首先贪心地寻找一个$(K,2K)$-Set-Covering 的可行解作为 K-Set-Covering 的近似最优解，再通过调整该解来获得最终的汇总表。

AlphaSum 的缺陷是：强制限定每个概化元组所概化的原始元组数均为 K，这不符合原始表的数据分布特征而造成更多的语义损失；K-Set-Covering 近似算法的引入使得汇总表的大小不为确定值而是介于($\lceil n/K \rceil$,$\lceil n/K \rceil$)之间。

效率方面，AlphaSum 转换的预处理开销大，需要计算且维护 $O(nh_{\text{mean}}^{m})$个概化元组，其中 h_{mean}是属性值概念分层的平均层数，m 是汇总属性个数。核心算法在最坏情况下的开销为 $\Omega(n^3 h_{\text{mean}}^{m})$。随着 h_{mean}与 m 的增加，核心阶段处理的数据量呈指数级增长，因此，AlphaSum 不适合高维属性上的语义抽象。

各种基于距离的空间聚类算法在文献[10]有详细介绍，文献[8,9]介绍了与本文聚类目标类似的空间聚类问题并提出了近似算法。

3 问题描述

本节对表语义压缩进行形式化定义，再将其建模为空间聚类问题。限于篇幅，本文省略数值型属性建立概念分层的方法(该方法在文献[5-7]均有说明)，假设所有被概化属性都已建立好概念分层，且分层中每条边对应的语义权重为 1。值得一提的是，本文算法同样适用于权重不同的情况。

3.1 符号约定和形式化定义

属性值的概念分层描述了属性值间的偏序关系，我们使用具有有向边的节点标签树模型来描述。设树 $ha(Va,Ea)$ 表示属性 a 的属性值概念分层，Va 为 ha 节点集，且与属性 a 的值域一一映射，函数 $label(x)$与 $level(x)$分别表示属性值 x 对应节点的标识(编码)及其在 ha 中的层数。Ea 为 ha 的有向边集，Ea 中任意有向边为一有序二元组，有向边 $e:\langle v_1,v_2\rangle$ 表示 ha 中存在一条从节点 v_1 到节点 v_2 的有向边；Ea^* 为 Ea 的传递闭包，它包含了 ha 中的所有路径，其描述的是 a 中任意属性值间的概化关系。集合 Va^{Inner} 和 Va^{Leaf} 分别代表 ha 中的内部节点集和叶节点集。

根据上述约定，下面分别给出属性值概化及元组间概化等相关形式化定义。

定义 1(语义量) 属性 a 中任意属性值 x 的语义量为 x 在概念分层中对应节点的层数，即 $level(x)$。元组的语义量为元组的所有属性值的语义量的累加和，即元组 $t:\langle x^1,x^2,\cdots,x^m\rangle$的语义量 $t.Semantic=\sum_{i=1}^{m} level(x^i)$。

定义 2(属性值间概化关系与元组间概化关系) 给定属性 a 的任意两个属性值 x_1 与 x_2，若 Ea^* 中存在$\langle label(x_1),label(x_2)\rangle$，则称 x_1 可以被概化到 x_2 或 x_2 可以概化 x_1，用 $x_2 \angle x_1$ 表示。$x_2 \angle x_1$ 表明：在语义汇总过程中可以使用属性值 x_2 来替换属性值 x_1。给定任意两个元组 t_1 和 t_2，若对于任意第 i 个属性均有 $t_1[i] \angle t_2[i]$成立，则称 t_1 可以概化 t_2，用 $t_1 \angle t_2$ 表示。

定义 3(属性值的概化范围与元组的概化范围) 对于属性 a 的属性值 x，若 $label(x) \subset Va^{Inner}$，则 x 的概化范围为 $x.gen\_Range=\{x' \mid x \angle x', label(x') \subset Va^{leaf}\}$。$x.gen\_Range$ 在 ha 中反映为以 x 对应节点为根节点的子树的所有叶节点对应的属性值集。显然，概化元组的概化范围为其所有属性值的概化范围的笛卡尔积。

定义 4(属性值集的最优概化属性值) 给定 a 的属性值 x 和任意属性值的集合 Sa，若满足条件：

(i) $Sa \subset x.gen\_Range$，

(ii) 不存在 x'，$Sa \subset x'.gen\_Range$ 且 $|x'.gen\_Range| < |x.gen\_Range|$，

则称 x 为 Sa 的最优概化属性值，表示为 $x \angle_{opt} Sa$。

性质 1 最优概化属性值在能概化给定属性值集的所有概化属性值中具有最大的语义量。

证明(反证法)：给定属性值集 Sa 及其最优概化属性值 x_1，假设存在概化元组 x_2，有 $Sa \subset x_2.gen\_Range$ 且 $level(x_1) < level(x_2)$。任取 Sa 中属性值 x'，由定义可知，$x_1 \angle x'$ 且 $x_2 \angle x'$，即 Ea^* 中存在路径 $p1=\langle label(x'),label(x_1)\rangle$和路径 $p2=\langle label(x'),label(x_2)\rangle$。又因为 $level(x_1) < level(x_2)$，故 Ea^* 中存在路径 $p3=\langle label(x_2),label(x_1)\rangle$。$p3$ 表明 $label(x_2)$是以 $label(x_1)$为根节点的子树中的节点，因此 $x_2.gen\_Range \subset x_1.gen\_Range$，则与定义 4 矛盾。

定义 5(元组集的最优概化元组) 若概化元组 tc 的每个属性值均是元组集 Ts 中所有元组对应属性值集的最优概化属性值，则称 tc 是 Ts 的最优概化元组，表示为 $tc \angle_{opt} Ts$。

性质 2 最优概化元组是所有能概化给定元组集的概化元组中具有最大语义量的概化元组。

证明(反证法)：由定义 5 可知，最优概化元组的任意属性值为(元组集在该属性上投影所得属性值集的)最优概化属性值。假设存在给定元组集的某概化元组具有比最优概化元组更大的语义量，则该概化元组必存在某个属性值上的语义量大于最优概化元组对应的属性值，即存在比最优概化属性值的语义量还大的属性值，这与性质 1 矛盾，故假设不成立。

3.2 语义质量度量

汇总表的语义质量度量是反映汇总表语义信息的标准。作为目标代价函数，它用于指导语义汇总表的构造过程。

语义度量准则 $Info\_Loss$ 计算所有原始元组概化时产生

语义损失的累加。$Info\_Loss$ 越小，则语义汇总表的质量越好，其计算公式如下：

$$Info\_Loss(T^{Summary})=\sum_{t\subset T}(t.Semantic-T^{Summary}(t).Semantic)$$

式中，$T^{Summary}$ 表示语义汇总表，函数 $T^{Summary}(t)$ 表示原始元组 t（在汇总表中）被替换的概化元组。

3.3 K-Summary 的形式化定义

给定表 T 和属性值的概念分层集 H、语义表的缩减率 K，构建满足如下条件的语义汇总表 $T^{Summary}$：

1) $|T^{Summary}|=\lceil |T|/K \rceil$；

2) $\bigcup_{ti\subset T^{Summary}} ti.gen\_Range=T$；

3) $Info\_loss(T^{Summary})$ 取得极小值。

4 基于层次聚类的 K-Summary 算法

通过引入 Dewey 编码，K-Summary 可被建模为空间聚类问题，4.1 节将对此进行讨论。4.2 节介绍如何利用 Deway 码将概念分层中的概化关系编码到原始数据，以提高语义量的计算效率。4.3 节介绍如何将 K-Summary 转换为 CMSP，即最小化分簇子空间周长之和；提出快速层次聚类算法和基于分辨率调整策略的算法。

4.1 将 K-Summary 转换为 d^p 空间聚类

由性质 2 可知，一旦划分确定，使用每个分簇的最优概化元组来替换分簇中的原始元组，可保证 $T^{Summary}$ 获得最大语义量。因此，$T^{Summary}$ 的质量主要依赖于原始表的划分。与传统线性空间不同，K-Summary 问题中元组所处空间为多维层次空间，其任意维度仅具有偏序关系而非良序。我们通过定义多维层次度量空间 d^p，将 K-Summary 转换为 d^p 空间的聚类问题。

定义 6（点间度量 d^p） T 上的映射 $d^p: T\times T->\mathbf{R}$，$\mathbf{R}$ 为实数集。任意 t_1 和 t_2 属于 T，点间距离 $d^p(t_1,t_2)=t.Semantic$，其中 $t\angle_{opt}\{t_1,t_2\}$。

显然，d^p 满足：(1)正定性；(2)对称性；(3)三角不等式。因此，d^p 是 T 上的一个度量。

定义 7（外接子空间、最小外接子空间及子空间周长 Φ^c） 任意 S 属于 2^T 且 $ts\angle S$，则 ts 为 S 中所有点的外接子空间。若 $ts\angle_{opt}S$，则为最小外接子空间，子空间周长 $\Phi^c(ts)=ts.Semantic$；若 $ts\angle ts'$，则称子空间 ts 包容子空间 ts'。

性质 3 d^p 空间中，对于任意 $i:[1,y]$，若 ts_i 是点集 S_i 的最小外接子空间且 ts 是点集 $\bigcup_{i=1}^{y}s_i$ 的最小外接子空间，则 ts 能包容 ts_i。

证明：由 $ts\angle_{opt}\bigcup_{i=1}^{y}s_i$ 可得：对于任意 $i:[1,y]$，有 $ts\angle s_i$。又因为 $ts_i\angle_{opt}s_i$，根据最小外接子空间定义可知 $ts\angle ts_i$，即 ts 能包容 ts_i。

定义 8（点集的子空间冗余度 R） 给定点集 s 及其外接空间 ts，则 s 在 ts 上的空间冗余量 $R(s,ts)=\frac{1}{|s|}\sum_{t\in s}(t.Semantic-ts.Semantic)$。

定义 9（分辨率 Resolution） d^p 空间中任意维具有偏序关系的层次结构，但其仍具有类似多维线性空间的分辨率概念。在 d^p 空间中分辨率由各维对应的层次结构的深度所决定。我们用函数 $Resolution(ts,g,m)$ 表示子空间 ts 按照策略 g 调整了 m 次后的结果，则 $Resolution$ 具有如下性质。

性质 4 设 ts_1、ts_2 和 ts_3 均为 d^p 中的子空间，给定调整策略 g，则：

1)若 $ts_1\angle ts_2$，则对于任意有效 x，$Resolution(ts_1,g,x)\angle Resolution(ts_2,g,x)$；

2)若 $Resolution(ts_1,g,n_1)==Resolution(ts_2,g,n_1)$ 且 $Resolution(ts_2,g,n_2)==Resolution(ts_3,g,n_2)$，则有 $Resolution(ts_1,g,\max\{n_1,n_2\})==Resolution(ts_3,g,\max\{n_1,n_2\})$。

上述性质表明，d^p 空间中相同的分辨率调整操作不会影响子空间间的包容关系。

根据上述定义，我们可将 K-Summary 问题做如下描述：

最小化外接子空间周长之和的聚类：在度量空间 d^p 中，寻找 B 个子空间，点集 T 中的任意点必包容于这 B 个子空间，其中一个且使得 B 个子空间周长之和最小。其中，$B=\lceil n/K \rceil$。

4.2 基于 Dewey 编码的快速计算

Dewey 编码是一种快速树编码，常用于树的索引及检索。其编码规则描述为：根节点编码为“0”；若节点 i 的编码为“0…xx”，则节点 i 的第 t 个子节点的编码为“0…xxt”。如图 1 所示，Age 和 Location 概念分层中每个节点的右上角数字为其 Dewey 编码。

Dewey 编码将分层关系编码到节点，通过节点的 Dewey 编码能快速判断节点间的概化关系。以属性值的概念分层的 Dewey 编码作为数据字典，将原始元组转换为度量空间 d^p 上的点，能快速计算点间距离、分簇外接空间周长以及冗余量等。表 1－表 4 中附加列 DeweyCode 记录了（以图 1 为数据字典的）元组 Dewey 码。

元组的语义量等于其对应 Dewey 码的长度，如表 1 中 $t_1.Semantic=|011|+|0111|=5$。由此做类似计算可得，表 4 的语义量分别为 8、7 和 4，因此表 4 的语义损失最小。给定元组集的 Dewey 码集，则该元组集上最优概化元组的 Dewey 码由 Dewey 码集在各维属性上投影所产生的字符串集的最长公共前缀子串组成。下面给出使用 Dewey 码来计算点集 $\{t_2,t_3\}$ 和 $\{t_1,t_2,t_3\}$ 的最小外接子空间及周长、点间距离等概念的过程。如下所示，分别计算出两个点集合的最小外接子空间 t_{23} 和 t_{123} 的 DeweyCode。

t_{23}		Age	Location	t_{123}		Age	Location
					t_1	011	0112
$\angle_{opt}$	t_2	012	0122	$\angle_{opt}$	t_2	012	0122
	t_3	011	0121		t_3	011	0121
	t_{23}	01	012		t_{123}	01	01

t_2 和 t_3 距离 $d^p(t_2,t_3)=t_{23}.Semantic=|Dewey(t_{23})|=2+3=6$；点集 $\{t_1,t_2,t_3\}$ 的最小外接子空间的周长 $\Phi^c(t_{123})=t_{123}.Semantic=|Dewey(t_{123})|=2+2=4$；通过编码可判断，子空间 $t_{123}\angle t_{23}$，即 t_{123} 包容 t_{23}；$\{t_2,t_3\}$ 在其外接子空间 t_{23} 中的空间冗余度 $R(\{t_2,t_3\},t_{23})=t_2.Semantic+t_3.Semantic-2*t_{23}.Semantic=7+7-2*5=4$。

4.3 基于层次凝聚的空间聚类算法

寻找（或构建）满足要求的 B 个子空间是上述空间聚类问题的关键，我们将采取两种思路来产生候选的子空间。初始时，将集合 T 中的 n 个点视为 n 个仅包含自身的点集，则这 n 个点集的最小外接子空间编码为其本身，集合 T 可看作

包含 n 个子空间的子空间集。第一种思路：新产生的候选子空间是由当前子空间集中某两个子空间合并后所产生的，每轮循环后，当前子空间集将减少一个(或者多个)子空间，如此反复直到满足要求；第二种思路：适当调整多维层次空间的分辨率，观察子空间集在低分辨率情况下的聚合情况，寻找适当的分辨率，使得 T 在该分辨率下聚合到 B 个子空间之中。

4.3.1 第一种思路设计的算法

如前所述，根据第一种思路设计的最简单的算法如算法1所描述。

算法 1 TSHC(T,K)

```
Input: T,set 原始数据的编码集;
       K,int 压缩率
Output:Rset,set 子空间编码集;
Begin
1. Rset:=Initial(T);
2. Cset:=Pairwise_V1(Rset);
3. While(|Rset|> |T|/K )
4.    Selitem:=Search_V1(Cset);
5.    RUpdate_V1(Rset,Selitem);
6.    CUpdate_V1(Cset,Selitem);
7. End While
8. Return Rset;
End
```

Rset 为子空间的集合，记录每轮处理后的 d^p 空间中子空间的分布情况。Rset 的成员对象为二元组(C,SCode)，C 和 Scode 分别记录当前子空间所包含的点集(的 id)和子空间的 Dewey 编码，其初始值为原始点的 id 和编码，初始化工作在 Initial 函数中完成。Cset 为每轮产生的候选子空间的集合。初始时，它由 Pairwise_V1 函数根据 Rset 进行两两构造所产生，且(在每轮循环中)由 CUpdate_V1 所维护。Cset 的成员对象为(RS,Scode,DeltaR)，集合 RS 记录本候选子空间所能包容的 Rset 中的子空间(算法 1 中 RS 仅保留产生该候选子空间的 2 个子空间)，Scode 为当前候选子空间的编码，DeltaR 保留将 RS 中子空间合并成本候选子空间将会产生的冗余量变化值。给定候选子空间 cs，$DeltaR(cs)=$

$$\frac{\sum_{rs\in cs.RS}|rs.C|R(rs.C,rs.SCode)}{\left|\bigcup_{rs\in cs.RS}rs.C\right|}-R\left(\bigcup_{rs\in cs.RS}rs.C,cs.SCode\right)$$

函数Search_V1 在 Cset 中寻找 DletaR 最小的候选子空间，并保留在 Selitem 中。函数 RUpdate 更新 Rset：根据 Selitem 构建新的子空间并加入 Rset，删除被合并的子空间。CUpdate_V1 更新 Cset：删除无效的候选子空间，即 Selitem. RS 中的子空间所产生的候选子空间；加入新产生的候选子空间，即 Rset 中新加入的子空间与原有子空间所产生的多个候选子空间。当 Rset 中成员个数满足目标后，算法结束。

输入的规模为 n 且输出的规模为 b 时，Cset 的规模为 $O(n^2)$，则核心操作 Search 的代价为 $O(n^2)$。若要为 Cset 建立红黑树作为二叉搜索树，并考虑维护代价可将查找代价降低为 $O(\lg n)$，则算法需迭代 $(n-b)$ 次，故算法的执行代价为 $O(n\lg n)$。

算法 1 的问题是 Rset 的收敛速度缓慢，每轮迭代 Rset 的子空间数仅减少一个。为了提高算法的效率，在确定 Selitem 时可加入对 Cset 中成员的包容能力(即 Selitem. RS 中子空间的个数)的考量。包容能力越高，Rset 中子空间数递减的速度越快。优先选择“包容系数”高的膨胀子空间，使得每轮迭代能合并 C 中更多的子空间来减少迭代次数，从而加快 C 的收敛速度。因此，优化的算法在确定 Selitem 时，不仅要考虑 DletaR，还要考虑候选子空间的包容能力。算法 2 对此进行了描述。

算法 2 TSHS_Quick(T,K,Threshold)

```
Input:T,relation 原始数据编码;
      K,int 压缩表大小;
      Threshold,float 阈值
Output:Rset,set 子空间编码集;
Begin
1. Rset:=Initial(T);
2. Cset:=Pairwise_v2 (Rset);
3. While(|C|> |T|/K )
4.    Citems:=PreSearch (Cset,Threshold);
5.    if(|Citems |==1)
6.       Selitem= Citems [0],Goto 8;
7.    Selitem =Search_v2(Citems);
8.    RUpdate_v2(Rset,Selitem);
9.    Update_v2(Cset,Selitem);
10. End While
11. Return Rset;
End
```

算法 2 中，Pairwise_V2 除了要完成算法 1 中 Pairwise_V1 的工作外，还要为每个候选子空间统计其所能包容的当前子空间，并保存于 RS(而 Pairwise_V1 仅保存生成的两个子空间)。PreSearch 对 Cset 进行预选工作，它不仅选择 DeltaR 获得最小值的候选子空间，还选择 DeltaR 在最小值的(1+Threshold)倍范围内的候选子空间，并将其保存于集合 Citems 中。Search_v2 在 Citems 中选择包容能力最大的候选子空间，即|RS|取得最大值的候选子空间。RUpdate_v2 与 CUpdate_v2 工作与算法 1 类似，不再赘述。

算法 2 比算法 1 更灵活，通过 Threshold 可控制算法的收敛速度。当 Threshold 为 0 时，算法 2 仍要快于算法 1。真实数据集上的实验表明，算法 2 的执行时间要少于算法 1。

4.3.2 第二种思路设计的算法

根据性质 4 可知，调整 d^p 空间的分辨率不会改变子空间的包容关系，降低分辨率会导致更多的子空间聚合到更大的子空间中去。因此，首先全局地逐步降低分辨率并观察子空间集在不同分辨率下的分布情况，找到最接近目标子空间数量的分布，再以此为基础适当地局部放大某些子空间的分辨率，来分裂这些子空间，直到子空间数达到目标要求。

使用 Dewey 编码可快速调整空间分辨率，调整分辨率操作实际上是规定 Dewey 编码各维的位长。例如，2 维空间在分辨率为(5,5)时，某两个子空间编码为 $ts_1=\langle 12345,11234\rangle$ 和 $ts_2=\langle 12346,11235\rangle$，若 $g_1=$“各维度降 1”即分辨率变为(4,4)，则 $Resolution(ts_1,g_1,1)==Resolution(ts_2,g_1,1)==\langle 1234,1123\rangle$。

如何合理设定分辨率调整策略，是可进一步优化的方向。例如，根据各子空间在各维度的频繁度(差异度)等统计参数来确定分辨率的调整方式。限于篇幅，本文仅采取基本的调整方式，即每轮各维度均降低一级，再局部放大分辨率。算法 3 对此进行了描述。

算法 3 ResAdjust(T,K)

```
Input: T,set 原始数据的编码集;
       K,int 压缩率
Output:Rset,set 子空间编码集;
Begin
1. Rset:=Initial(T);
2. Tmpset:=Null;
3. While(|Rset|> |T|/K )
4.   Tmpset:=Null;
5.   Foreach(rs in Rset)
6.     trs=Resolution(rs,g,1);
7.     Insert(tmpSet,trs);
8.   End Foreah
9.   Rset=tmpSet;
10. End While
11. While(|Rset|! = |T|/K)
12.     Set1=Null,Set2=Null;
13.     Tmpset=Selet(Rset);
14.     Delete(Rset,Tmpset);
15.     Split(Tmpset,&Set1,&Set2);
16.     Insert(Rset,Set1,Set2;)
17. End While
18. Slim(Rset);
19. Return Rset;
End
```

集合 Rset 记录分辨率调整后 d^p 空间中的子空间分布情况。行 3 至行 10 采取每轮“各维降低 1”的策略来全局降低分辨率。依次对 Rset 中的子空间 rs 使用 Resolution 函数降低分辨率，并使用 Insert 函数插入到临时子空间集 tmpSet 中，Insert 还要维护(高分辨率的)Rset 到(低分辨率的)tmpSet 中的子空间映射关系。行 11 到行 17 对 Rset 中的部分子空间进行局部分辨率放大。每次通过 Selet 函数选择放大 Rset 中某个子空间的分辨率，使得该子空间分裂成两个子空间，且获得最大的空间冗余量变化 DeltaR，函数 Delete，Split 和 Insert 完成对选定子空间的分裂以及维护 Rset 的更新工作。如此反复，直到 Rset 集合中子空间数满足目标。行 18 的 Slim 函数对 Rset 中每个子空间所包含的原始点重新计算最小外接子空间，以此来降低分辨率过度调整所带来的冗余。

显然，算法 3 在执行速度上要远快于算法 2。算法 3 在降低空间分辨率时，考虑的是子空间的冗余量的定量变化。粗放地对子空间进行降低分辨率的操作，会使得更多的子空间在算法初期被合并。一般来说，越多的子空间在早期被合并，则其产生的冗余量变化就会越少。粗放的调整也会产生大量不必要的冗余量，虽然算法 3 后期又以此为基础局部提高部分子空间的分辨率，但在多数情况下，算法 2 能比算法 3 得到更高质量的解。

在实际应用中，对于大规模的数据，可以使用算法 3 作为处理的前端来减小数据规模，再使用算法 2 获得质量较好的解，从而获得速度与质量的平衡。

5 实验与分析

5.1 实验环境

实验采用主频为 1.6GHz 的 CPU(Intel CoreDuo E2140)以及 1GRAM(DDR)；算法使用 CSharp 实现，编译环境为 VS.NET2005。

实验采用真实数据集 Adult Dataset 作为实验的数据，该数据集与文献[4]采用数据集相同。我们消除了数据集中缺失值的元组并选择 5 个属性作为实验对象。实验数据集中约含有 45k 条元组，其属性的特性描述如表 5 所列。

表 5 AdultDataSet 实验对象属性特性描述

属性名	属性值个数	属性值概念分层数
Age	74	4
Work Class	7	3
Education	16	4
Martial Status	7	3
Native Country	41	3

实验从执行效率和结果质量两方面来考量本文算法，并与经典的语义压缩算法 AlphaSum 进行对比。结果质量以压缩前后的语义损失率 *ILR* 作为度量标准，其计算公式如下：

$$ILR(T^{\text{Summary}})=\frac{Info\_Loss(T^{\text{Summary}})}{\text{T. Semantic}}\times 100\%$$

式中，T. Semantic 为原始数据的语义量。

5.2 真实数据集上的实验结果与分析

我们在 Adult 数据集中选择具有不同维数的 3 组数据进行实验，在不同压缩率情况下对 TSHS_Quick，ResAdujst 和 AlphaSum 进行考察。实验的目的是观察数据集维度和压缩率的变化对算法效率及质量所产生的影响。实验结果如图 3 至图 8 所示。

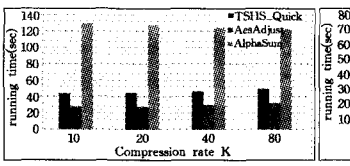

图 3 属性个数 $m=3$ 的算法运行时间实验结果

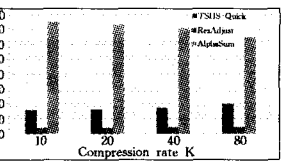

图 4 属性个数 $m=4$ 的算法运行时间实验结果

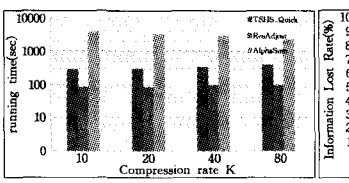

图 5 属性个数 $m=5$ 的算法运行时间实验结果

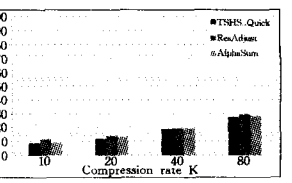

图 6 属性个数 $m=3$ 的算法压缩质量实验结果

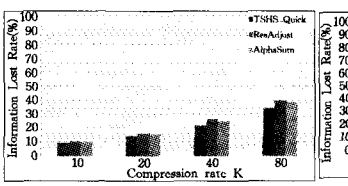

图 7 属性个数 $m=4$ 的算法压缩质量实验结果

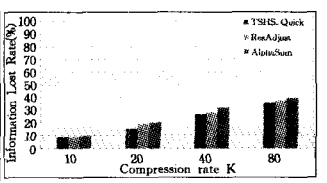

图 8 属性个数 $m=5$ 的算法压缩质量实验结果

如图 3—图 5 所示，TSHS_Quick 和 ResAdjust 算法的运行效率相对于经典算法 AlphaSum 有大幅提高。在 3 组数据中，算法 ResAdjust 具有最短的运行时间，TSHS_Quick 次之，AlphaSum 运行时间最长。就运行时间而言，压缩率 K 对 3 种算法的影响较小，而被压缩属性的个数 m 对 3 种算法的影响较大。就被压缩属性的个数 m 而言，ResAdjust 受到的影响最小，TSHS_Quick 次之，AlphaSum 最大。因此，将 *K*-

Summary 作为空间聚类问题处理能获得更好的运行效率，并且经典算法 AlphaSum 不适合高维数据。如图 6—图 8 所示，TSHS_Quick 算法产生的实验结果具有最好的质量，而 Res-Adjust 和 AlphaSum 的结果质量不相上下。就结果质量而言，3 种算法中，K 的影响要大于 m 的影响。因此，将 K-Summary 作为空间聚类问题处理，能获得更好质量的语义压缩表。

结束语 通过使用 Dewey 编码将原始元组编码成多维层次空间中的点，本文将 K-Summary 问题转换为 d^p 空间上的以最小化外接子空间周长之和为目标的空间聚类问题。基于凝聚策略和分辨率调整策略，我们提出了两种解决该空间聚类的算法。相对于现有的表语义汇总系统，本文算法更高效且能保留更多的语义信息。

下一步的工作是解决不确定数据表的语义汇总问题。数据属性值的不确定性使得属性值在概念分层上具有多条通向根节点的路径，且每条路径具有不同的概率值。如何在概率模型下解决 K-Summary 问题，并给出其结果的概率语义解释，是今后的研究重点。

参 考 文 献

[1] Lo M-L, Wu K-L, Yu P S. Tabsum: A flexible and dynamic table summarization approach[C]// ICDCS. 2000: 628

[2] Paul S R, Raschia G, Mouaddib N. Database summarization: The saintetiq system[C]// ICDE. 2007: 1475-1476

[3] Paul S R, Raschia G, Mouaddib N. General purpose database summarization[C]// VLDB. 2005

[4] Candan K S, Cao Hui-ping, Qi Yan. AlphaSum: Size-constrained Table Summarization Using Value Lattice[C]// EDBT. 2009

[5] Byun J-W, Kamra A, Bertino E. Efficient k-Anonymization Using Clustering Techniques[C]// DASFAA. 2007

[6] Li Jiu-yong, Raymond C W, Pei Jian. Achieving k-Anonymity by Clustering in Attribute Hierarchial Structures[C]// DAWAK. 2006

[7] Li Jiu-yong, Raymond C W, Pei Jian. Anonymization by Local Recoding in Data with Attribute Hierarchical Taxonomies[J]. TKDE, 2008, 20(9)

[8] Bilo V, Caragiannis I, Kaklamanis C, et al. Geometric clustering to minimize the sum of cluster sizes[C]// Proceedings of the European Symposium on Algorithms. LNCS vol 3669, 2005: 460-471

[9] Charikar M, Panigrahy R. Clustering to minimize the sum of cluster diameters[J]. Journal of Computer and Systems Sciences, 2004, 68(2): 417-441

[10] Han Jia-wei, Kamber M. 数据挖掘概念与技术[M]. 范明，孟小峰，译. 北京：机械工业出版社，2001

[11] Noga A, Dana M, Shmuel S. Algorithmic construction of sets for k-restrictions[J]. ACM Trans. Algorithms, 2006, 2(2): 153-177

（上接第 152 页）

需再做处理，因此便捷性更高。此外，本文方法在查找频繁模式的同时，可同步高效地进行子串归并。

另一方面，在某些应用中，用户并不希望查找低频串。龚的方法可以剪枝低频串，提高速度，本文方法则不能做到。本文方法与其他主要方法的特性比较如表 1 所列。

表 1 算法特性比较

算法	语料大于内存	基于内存	并行处理	同频子串归并	适用大字符集
后缀数组＋Yamato	×	√	×	×	×
Hunt	√	×	×	×	×
Schurmann	√	×	×	×	×
Chen	√	√	√	×	√
龚	√	√	√	×	√
本文算法	√	√	√	√	√

结束语 为处理大规模语料，本文提出了基于语料划分的频繁模式查找算法。其将语料按后缀首字符划分为多个集合，通过集合中的最大化最长公共前缀区间(MLCPI)查找出频繁模式及其频次，所有集合查找结果的并集即为语料的频繁模式集合。查找时，根据具体需要可归并子串。试验表明，本文算法查找过程内存消耗稳定，受语料规模影响较小，4.61G纯文本语料内存消耗小于 30M，可处理规模远大于内存的语料。算法容易实现并行处理，从而获取更快的查找速度。此外，同步归并子串给频繁模式查找带来的时空影响较小。

查找出的频繁模式作为新词识别的候选词可提高识别的召回率，后续工作将围绕新词识别展开。

参 考 文 献

[1] Hunt E, Atkinson M P, W I R. A database index to large biological sequences[C]// Proc. of the 27th VLDB Conf. Roma, Italy: Springer, 2001: 139-148

[2] 邹纲. 中文新词语自动检测研究[D]. 北京：中国科学院研究生院，2004

[3] Ukkonen E. On-line construction of suffix trees[J]. Algorithmica, 1995, 14

[4] Manber U, Myers G. Suffix arrays: A new method for on-line string searches. [J]. SIAM J. Comput., 1993, 22(5): 935-948

[5] Yamamoto M, Churcht K W. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus[J]. Computational Linguistics, 2001, 27(1): 1-30

[6] Larsson N J, Sadakane K. Fast suffix sorting [R]. Sweden: Dept. of Computer Science, Lund University, 1999

[7] Juha Karkkainen P S. Linear Work Suffix Array Construction [J]. Journal of the ACM, 2006, 53(6): 918-936

[8] Klaus-Bernd S, Stoye J. Suffix Tree Construction and Storage with Limited Main Memory [R]. Germany: University of Bielefeld, 2003

[9] Chen Z, Fowler R. Fast construction of generalized suffix trees over a very large alphabet[C]// Proc. of Int Conf on Computing and Cobinatorics. 2003

[10] 龚才春，贺敏，陈海强，等. 大规模语料的频繁模式快速发现算法[J]. 通信学报，2007, 28(12): 161-166

[11] 吕学强，张乐，黄志丹，等. 基于散列技术的快速子串归并算法[J]. 复旦学报：自然科学版，2004, 43(5)

[12] Lü Xue-qiang, Zhang Le, Hu Jun-feng, et al. Statistical Substring Reduction in Linear Time[C]// Proc of the Conf First Int Joint Conf on Natural Language Processing. Sanya, Hainan Island, China: ACL, 2004: 320-327