基于大规模语料划分的频繁模式查找算法

丁溪源[1,3] 黄河燕[2] 张海军[3] 王树梅[1]
(南京理工大学计算机科学与技术学院 南京 210094)[1]
(北京理工大学计算机科学技术学院 北京 100081)[2]
(中国科学院计算机语言信息工程研究中心 北京 100097)[3]

摘 要 频繁模式查找对新词识别、网络舆情监测、生物信息序列检测等领域有很高的应用价值。为处理规模远超出内存的语料,提出了一种实用的频繁模式查找算法。先将语料按后缀首字符划分为多个集合,通过逐条扫描集合数据,搜索出最大化最长公共前缀区间(MLCPI)来完成查找。另外在此基础上提出逐层归并算法,实现查找的同时归并子串。由于进行查找时无需将全部数据导入内存,因此资源消耗较少;各集合间频繁模式查找互不干扰,可采用并行处理加快运行速度。使用4.61G纯文本语料进行了试验,结果表明其内存消耗小于30M,查找速度最快达1.08M/s,能高效地进行子串归并。

关键词 频繁模式,重复串,语料划分,子串归并
中图法分类号 TP391 **文献标识码** A

Algorithm of Frequent Patterns Finding Based on Large Scale Corpus Partition

DING Xi-yuan[1,3] HUANG He-yan[2] ZHANG Hai-jun[3] WANG Shu-mei[1]
(School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China)[1]
(School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)[2]
(Research Centre of Computer & Language Information Engineering, Chinese Academy of Sciences, Beijing 100097, China)[3]

Abstract Frequent patterns finding is useful for some areas, such as new word recognition, internet public opinion monitoring, bio-information series detection, etc. Considering that corpus size is much larger than memory capacity, we put forward a pragmatic algorithm to find frequent patterns. Firstly, corpus was partitioned into multiple sets based on first character of suffix, and then a concept of maximized longest common prefix interval (MLCPI) was introduced, and by means of searching it while scanning data in sets item by item, we accomplished the finding task. Besides, we proposed hierarchical reduction algorithm (HRA) to reduce sub-string during the finding process on that basis. There is no need to import all data into memory, so it will decrease resource consumption. Moreover, it is found that frequent patterns among sets do not interfere with each other, which will improve the speed while processing paralleled. We used 4.61 gigabytes plain text as experiment data. The results show that the memory usage is lower than 30 megabytes, and the speed is up to 1.08 megabytes per seconds, and it is able to reduce sub-string efficiently.

Keywords Frequent pattern, Repeats, Corpus partition, Sub-string reduction

1 引言

计算机网络的普及促进了语言的迅速发展。网络论坛、博客和即时通信等信息媒体中出现了大量新的词语、短语、非正式用语,如"灌水"、"打酱油"、"火星文"、"—_—!!"等。将这些频繁出现的用语快速查找出来,对新词识别、社会热点发现、网络舆情监测等有重要作用。此外,生物信息领域存在大量的生物信息数据,如DNA序列、蛋白质序列等,查找出这些信息中频繁出现的片段,有助于促进对生物进化、物种相关性等的研究[1]。但网络信息与生物信息数据不断迅速增长,其规模都远大于一般计算机内存,因此从庞大的数据中高效地查找出频繁模式是一项具有挑战性的工作。

本文提出一种针对大规模语料的频繁模式查找算法。首先将初始语料划分成多个集合,从各集合查找出频繁模式,各集合频繁模式的并集即为最终的查找结果。本文第2节介绍相关工作;第3节介绍本文算法及其实现;第4节介绍基于频繁模式查找的子串归并方法;最后是试验和结论。

2 相关工作

存在包含n个元素的序列$T=C_1,\cdots,C_n$,若序列$S=C_i$,

到稿日期:2011-04-02 返修日期:2011-06-30 本文受国家863计划重点项目(2006AA010109)资助。
丁溪源(1985—),男,硕士生,主要研究方向为新词抽取;黄河燕(1963—),女,研究员,博士生导师,主要研究方向为机器翻译及自然语言处理;张海军(1973—),男,博士生,讲师,主要研究方向为自然语言处理和新词识别;王树梅(1957—),女,博士,教授,CCF高级会员,主要研究方向为智能信息处理、数据库技术,E-mail:hwasm@mail.njust.edu.cn(通信作者)。

…，C_j，其中 $1\leqslant i<\cdots<j\leqslant n$，在 T 中出现的次数大于等于 2 次，则称 S 为频繁模式。从 T 中查找出 S，并统计其出现频次的过程，即为频繁模式查找。

传统的查找算法有 N 元递增算法、Sequitur 算法、基于后缀树和后缀数组的查找算法等。N 元递增算法的主要思想是依次查找语料中重复出现的二字串，…，N 字串。这种方法容易实现，但算法时间复杂度高，查找时间较长。Sequitur 算法在读取语料后生成产生式规则，再从规则中查找出频繁模式。这种方法速度较快，但查找时会遗漏某些频繁模式[2]。基于后缀树或后缀数组的方法，需要先读取语料，构造出后缀树或后缀数组，再从中查找出频繁模式。使用后缀树查找，建树时间较长，尽管 Ukkone[3] 等提出了线性时间的构造方法，但是总体而言内存消耗较大，往往是语料规模的数倍，且不适用于大字符集数据。Manber 和 Myers[4] 提出的后缀数组是后缀树很好的替代品，Yamato[5] 使用该结构查找频繁模式，不仅速度较快，而且大大地压缩了内存占用。文献[6,7]对后缀数组生成算法进行了改进，但所需内存仍是语料规模的数倍。

上述方法难以处理规模超出内存的语料。为此研究人员提出了其它解决方案，已有方法可分成两类：第一类主要基于磁盘。如 Hunt[1] 提出多次读取语料，将以某一前缀为首的所有后缀在磁盘上建立后缀树，其建树时间复杂度为 $O(n^2)$。Schurmann[8] 提出的 Clustered 算法改进了 Hunt 的方法，其时间复杂度为 $O(n\log n)$。这种方法需要磁盘空间较大，且多次读取语料，IO 操作频繁。第二类主要思路是将语料划分之后再作处理。如 Chen[9] 提出了一种针对大字符集的划分方法，即将语料中的后缀写入临时文件中，并对其进行排序，使得相同首字符的后缀在一个组，再将各组数据导入内存建树。龚才春[10] 提出按首字符对大语料中的后缀进行划分并保存到磁盘，再依次将划分后的结果导入到内存，建立倒排索引，从索引表中发现频繁模式且进行低频剪枝，加快查找速度。若划分后规模仍大于内存，则继续按第二个，…，第 N 个字符对后缀进行划分，直到后缀集合可以一次性导入内存为止。

3 基于语料划分的频繁模式查找

3.1 语料划分策略

设 Σ 是包含有限个字符的字符集，所有字串都由 Σ 中的字符构成。文本 T 由 1 个或多个字串组成，其开端、结尾以及字串间用分隔符“#”间隔。如 $T=\#ab\#abc\#$。

定义 1 若字串 S_1 和 S_2 的前 k 个字符都相同，第 $k+1$ 个字符不相同或者不存在，则称该 k 个字符组成的字串 S 为 S_1 和 S_2 的最长公共前缀，k 为 S 的长度。

上述定义仅针对两个字串。若扩展为以多个字串为研究对象，则有：

定义 2 在字串集合 $\Psi=\{S_1,\cdots,S_t\}$ 中，如果从 S_i 到 S_j $(1\leqslant i<j\leqslant t)$ 这 $j-i+1$ 个字串的前 k 个字符都相同，第 $k+1$ 个字符不相同或某一串长度小于 $k+1$，则称 $\langle i,j\rangle$ 为最长公共前缀区间。这 k 个字符组成的字串 S 为区间的最长公共前缀，记作 $LCP\langle i,j\rangle$。

定义 3 若定义 2 中集合 Ψ 的所有元素升序排列，即 $S_1\leqslant\cdots\leqslant S_t$。如果 $LCP\langle i,j\rangle$ 不为空字串，且 $LCP\langle i-1,j\rangle\neq LCP\langle i,j\rangle$ 与 $LCP\langle i,j+1\rangle\neq LCP\langle i,j\rangle$ 同时成立（若 S_{i-1}（或 S_{j+1}）不存在，则 $LCP\langle i-1,j\rangle$（或 $LCP\langle i,j+1\rangle$）为空字串），则称 $\langle i,j\rangle$ 是集合 Ψ 中的一个最大化最长公共前缀区间（Maximized Longest Common Prefix Interval，MLCPI），记作 $Max\langle i,j\rangle$。区间中所有字串的最长公共前缀为 $LCPMax\langle i,j\rangle$，集合 Ψ 中所有的 MLCPI 记为 $Max(\Psi)$。

例如，$T=\#ab\#abc\#abcdg\#abcdef\#abcdefg\#$，首字符为 a 的后缀组成集合 $\{ab,abc,abcdef,abcdefg,abcdg\}$，其中元素升序排列。$LCP\langle 1,5\rangle=ab$，$LCP\langle 2,6\rangle=$空，$LCP\langle 2,5\rangle=abc$，则 $\langle 2,5\rangle$ 是 MLCPI。

定义 4 在序列 $S=C_1\cdots C_n$ 中，从某位置 i 开始到末尾 n 结束的子串 $C_i\cdots C_n$ $(i\geqslant 1)$ 为 S 的后缀。

根据定义有如下性质。

性质 1 若 $LCP\langle i,j\rangle$ 存在，则其频次为 $j-i+1$；

性质 2 集合 $Max(\Psi)$ 中任意两元素 $Max\langle i,j\rangle$ 与 $Max\langle p,q\rangle$ 之间的关系为下述 3 者之一：

1) $Max\langle i,j\rangle\subseteq Max\langle p,q\rangle$，即 $p\leqslant i<j\leqslant q$；

2) $Max\langle p,q\rangle\subseteq Max\langle i,j\rangle$，即 $i\leqslant p<q\leqslant j$；

3) $Max\langle i,j\rangle\cap Max\langle p,q\rangle=\phi$，即 $j<p$ 或者 $q<i$。

性质 3 若 $\langle i,j\rangle\neq\langle p,q\rangle$，则 $LCP\langle i,j\rangle\neq LCP\langle p,q\rangle$。

分析定义和性质，得：

定理 1 文本 T 中，所有首字符为 ∂ 的后缀组成集合 $\Psi=\{S_1,\cdots,S_t\}$，若集合中元素升序排列，则 T 中首字符为 ∂ 的频繁模式集合为

$$\Omega=\{LCPM_1,\cdots,LCPM_i,\cdots,LCPM_n\}$$

式中，$M_i\in Max(\Psi)$，$Max(\Psi)$ 中元素数为 n。

同理可得其它首字符的频繁模式集合。因此文本 T 的频繁模式集合是以字符集中所有元素为首字符的频繁模式集合的并集。

证明：要证明定理 1，只需证明：

1) 集合 Ω 中的元素都是文本 T 中的频繁模式；

2) 文本 T 中的频繁模式都能在集合 Ω 中找到。

证明如下：

1) 若 $Max(\Psi)$ 不为空，对于任意的 $Max\langle i,j\rangle\in Max(\Psi)$，根据性质 1，$LCPMax\langle i,j\rangle$ 的频次为 $j-i+1>1$，因此 $LCPMax\langle i,j\rangle$ 是频繁模式且首字符为 ∂。$Max\langle i,j\rangle$ 中所有元素都在 T 中出现，因此这些元素的前缀也必能从 T 中找到，所以集合 Ω 中元素都为 T 中的频繁模式。得证。

2) 假设 T 中存在频次为 n 的频繁模式 R，其长度为 k，首字符为 ∂，则必能从 T 中找到且只能找到 n 个首字符为 ∂ 的后缀，它们的前 k 个字符相同，组成字串 R。根据定义，这些后缀都是集合 Ψ 的元素。由于元素升序排列且该 n 个字串前 k 个字符相同，则它们在集合中将会两两相邻，即为 S_i，…，S_{i+n-1}。

采用反证法证明 $\langle i,i+n-1\rangle$ 是 MLCPI。假设存在 $LCP\langle i-1,i\rangle$ 与 $LCP\langle i+n-1,i+n\rangle$，倘若 $LCP\langle i-1,i\rangle=R$ 或者 $LCP\langle i+n-1,i+n\rangle=R$，说明文本中前缀包含 R 的后缀数大于 n，这不符合前提条件。因此根据定义 3，$\langle i,i+n-1\rangle\in Max(\Psi)$，所以 $R\in\Omega$。得证。

3.2 频繁模式查找算法的实现

算法使用 $LCP\langle i-1,i\rangle$ 长度确定 MLCPI，其中 $LCP\langle i-1,i\rangle$ 表示当前读入串与前一串的最长公共前缀，其长度记为 $LCP[i]$，i 为当前读入串编号。当 i 为 0 时，$LCP\langle i-1,i\rangle$ 长

度为 0。集合末尾扩展一空串，其与前一串的最长公共前缀长度为 0。

例　从文本 $T=\#ab\#abc\#abcdg\#abcdef\#abcdefg\#$ 中查找频繁模式过程，见图 1。图中左侧表格表示以 a 为首字符的后缀信息，右侧是从各区间查找出的频繁模式及其频次。

序号	字串	LCP<i-1,i>长度
1	ab	0
2	abc	2
3	abcdef	3
4	abcdefg	6
5	abcdg	4
6	空	0

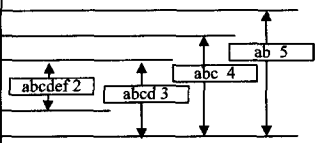

图 1　通过 MLCPI 查找集合中的频繁模式

算法定义 i_stack，j_stack 两个堆栈。

输入：语料后缀集合。

输出：频繁模式集合。

```
1 FOR i=1 To n //n是输入集合中后缀总条数;
2   IF LCP〈i-1,i〉长度 〈 LCP〈i,i+1〉长度 THEN
3     i入栈 i_stack,LCP〈i,i+1〉长度入栈 j_stack;
4   ELSEIF LCP〈i-1,i〉长度 〉 LCP〈i,i+1〉长度 THEN
5     k=j_stack 栈顶数据,j_stack 弹出顶端数据;
6     q=i_stack 栈顶元素;
7     输出第 i 条字串的前 k 个字符为频繁模式,频次是 i-q+1;
8     进入循环,j_stack 不为空与 LCP〈i,i+1〉长度<k 往下执行,
      否则跳到第 14 步;
9     i_stack 弹出顶端数据;
10    q=i_stack 栈顶元素;
11    k=j_stack 栈顶数据,j_stack 弹出顶端数据;
12    输出第 i 条字串的前 k 个字符为频繁模式,频次是 i-q+1;
13    j_stack 为空或 LCP〈i,i+1〉长度>=k 往下执行,否则跳到第
      8 步;
14    IF j_stack 为空 THEN
15      IF LCP〈i,i+1〉长度是 0 THEN
16        i_stack 弹出顶端数据;
17      ELSE
18        LCP〈i,i+1〉长度入栈 j_stack
19      ENDIF
20    ELSE
21      IF LCP〈i,i+1〉长度=j_stack 栈顶数据 THEN
22        i_stack 弹出顶端数据;
23      ELSEIF LCP〈i,i+1〉长度〉j_stack 栈顶数据 THEN
24        LCP〈i,i+1〉长度入栈 j_stack
25      ENDIF
26    ENDIF
27  ENDIF
28 ENDFOR
```

根据算法，从上例中文本 T 中查找出 $abcdef$ 的过程，见图 2。顺序扫描数据，$i=1$ 时，$LCP[1]<LCP[2]$，1 入栈 i_stack，$LCP[2]=2$ 入栈 j_stack 。同理，得 i 为 3 时的堆栈数据，见图 2 左侧。

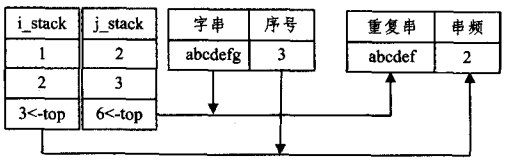

图 2　查找频繁模式 abcdef 的过程

当读入 $i=4$ 的字串时，$LCP[i+1]<LCP[i]$，且 $LCP[i+1]<j$_stack. top()，查找出 $abcdefg$ 的前 6 位字符为频繁模式 $abcdef$，次数为 $i-i$_stack. top()+1 即 2，弹出 j_stack. top()；同时 $LCP[i+1]>j$_stack. top()，将 $LCP[i+1]$压入堆栈 j_stack。

4　子串归并

定义 5　设 $S_1=C_i\cdots C_j$ 和 $S_2=C_p\cdots C_q$，若 $S_2\subset S_1$，即 $S_1=C_i\cdots C_p\cdots C_q\cdots C_j$，其中 $i=p$ 与 $j=q$ 不同时成立，则称 S_1 是 S_2 的父串，S_2 是 S_1 的子串。

定义 6　若定义 5 中 S_1 和 S_2 是同一文本中查找出的频繁模式且频次相同，则称剔除 S_2 的过程为子串归并。

在未登录词或新词识别等应用中，有时需将频繁模式进行子串归并后作为候选词。传统的算法，每判断一字串是否要归并，都需要搜索全部候选串。当归并前候选数为 n 时，时间复杂度为 $O(n^2)$，处理效率较低。吕学强[11]提出基于散列表的算法，将所有频繁模式及其频次存放于散列表中，依次判断表中每一项所有可能的子串是否存在。若存在且频次相同，则归并该子串，其时间复杂度为 $O(n)$。这种方法速度虽然较快，但需将数据全部存入散列表，当数据规模超出内存时，则无法处理。吕[12]还提出另一种 $O(n\log n)$的方法，该方法分两步：先将候选串排序，根据频次标记出需要归并的子串；第二步将候选串颠倒后再排序，标记出需要归并的子串。处理过程经历两次字串排序，一次顺序颠倒，IO 操作频繁，消耗时间较长。

上述方法都需在查找出频繁模式后单独进行归并。本文提出逐层归并算法（Hierarchical Reduction Algorithm，HRA），归并过程与频繁模式查找同步进行。

定理 2　若 Max$\langle i,j\rangle$中所有串在文本 T 中的前一个字符都相同，且这个相同字符不是分隔符“#”，则 LCPMax$\langle i,j\rangle$必为另一串的子串，且两串的频次都为 $j-i+1$。

证明：由于 Max$\langle i,j\rangle$集合中各字串都从 T 中查找，设区间中所有字串的前一字符相同，为 C。若 C 是“#”，说明 LCPMax$\langle i,j\rangle$的首字符出现在开端，必不为其他串的子串；若 C 不是“#”，则 LCPMax$\langle i,j\rangle$是 C+LCPMax$\langle i,j\rangle$的子串，且二者同现，每出现一次 LCPMax$\langle i,j\rangle$，必出现一次 C+LCPMax$\langle i,j\rangle$。根据性质 1，LCPMax$\langle i,j\rangle$的频次是 $j-i+1$，因此 $C+LCPMax\langle i,j\rangle$的频次也等于 j−i+1。

定理 3　设 S_1 和 S_2 是同一文本中查找出的频繁模式，S_2 是 S_1 的子串，即 $S_1=C_i\cdots C_xC_p\cdots C_q\cdots C_j$，$S_2=C_p\cdots C_q$，其中 $i\neq x$。若 S_2 与 $S=C_xC_p\cdots C_q$ 的频次不相同，则其与 S_1 的频次也不相同。

证明：若 S_2 与 S 的频次不相同，由于 S_2 是 S 的子串，因此 S_2 的频次必大于 S 的频次。又由于 S 是 S_1 的子串，因此 S 的频次必大于等于 S_1 的频次，所以 S_2 的频次必大于 S_1 的频次。

在定义 5 中，若 S_2 是 S_1 的子串，有 3 种可能：1）$i=p$，$j\neq q$；2）$i\neq p$，$j\neq q$；3）$i\neq p$，$j=q$。

由于频繁模式查找以最长公共前缀为依据，根据性质 1 和性质 3，若同一集合中查找出的频繁模式不相同，则其频次亦不相同。所以在第 1 种可能下，S_1 和 S_2 频次必不相同，无需归并 S_2。

在第 2 和第 3 种可能下，根据定理 3，若 S_2 不与 C_x+S_2 频次相同，其中 C_x 是 Σ 中任意字符，则 S_2 必不与任何包含 C_x+S_2 的父串频次相同。因此根据定理 2 可逐层地进行归并，达到最终归并的效果。

5 试验及数据分析

5.1 试验环境

算法采用 C++实现。计算机配置为 Intel 酷睿 2Q9400，主频为 2.66GHz 的 4 核处理器，2G 内存。采用搜狗实验室提供的 124G 网页解析后得到的 4.61G 纯文本为试验数据。

5.2 试验数据分析

为测试算法性能，进行了 3 组试验。

第一组：将 4.61G 纯文本数据大致分成 5 等份，第 i 次试验采用其中的 i 份数据，同步归并与不归并子串各进行 5 次查找试验。算法运行内存情况如图 3 所示。

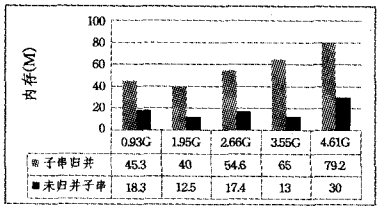

图 3 归并子串与否内存使用情况

图 3 表明，本文算法内存消耗比较稳定。处理数据小于 4.61G 时，不归并子串使用内存不高于 30M，同步归并则小于 80M。由于在查找时对集合数据进行扫描操作，无需将所有数据导入到内存，因此内存消耗受语料规模影响较小。图 3 亦表明，归并对频繁模式查找的空间消耗较小，处理 4.61G 语料时仅比不进行归并多占用 49.2M 内存。

随着语料规模的增加，查找的复杂程度亦会增长，查找时间如图 4 所示。

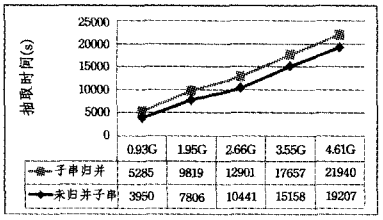

图 4 归并子串与否频繁模式查找时间

图 4 表明，同步归并子串与否，算法时间都随着语料规模的增加而缓慢增长，二者基本呈线性关系；且同步归并对查找时间影响较小。从图可以看出，二者之间的时间差基本保持恒定。

第二组：以首字符为语料集合划分依据，各个集合之间没有交集，因此可并行处理，以加快查找速度。采用多线程对 0.93G语料进行 12 次试验，第 i 次试验将字符集元素平均分成 i 份，使用 i 个线程并行处理，每个线程处理其中 1 份。查找时间见图 5。

试验结果表明，随着线程数的增加，查找速度显著加快。由于每个线程处理的数据量不相等，因此并非设置与处理器内核数相同多的线程数就能达到最优效果。上图表明 4 核处理器条件下，当线程数达到 8 个后，查找时间趋于平稳，0.93G纯文本语料查找速度最快达 1.08M/s。

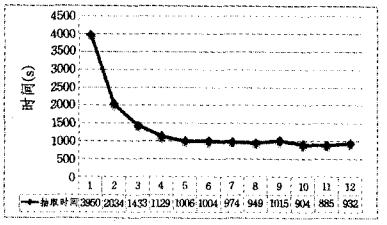

图 5 0.93G 语料多线程查找时间比较图

第三组：为检测逐层归并算法的有效性，定义 $\Delta add = MT - NMT$，其中 Δadd 是同步归并子串带来的额外时间，MT 是同步归并子串时的频繁模式查找时间，NMT 是不归并子串时的查找时间。设 $\Delta reduction$ 为查找出频繁模式后再做子串归并处理的时间。相同条件下，若 Δadd 远大于 $\Delta reduction$，则说明算法有效性较差。试验步骤为：

1）首先不归并查找出频繁模式，获得 NMT；

2）采用文献[11]中 $O(n)$算法对查找结果归并子串，获得 $\Delta reduction$；

3）再测试同步归并子串查找时间，获取 MT，得到 $\Delta add = MT - NMT$。

将 200M 纯文本数据大致分成 5 等份，第 i 次试验处理其中的 i 份数据，进行 5 次试验。Δadd 与 $\Delta reduction$ 对比结果如图 6 所示。

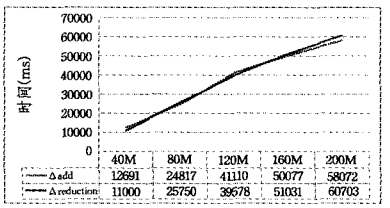

图 6 归并时间比较

图 6 表明，同等条件下 Δadd 与 $\Delta reduction$ 较接近，即同步进行子串归并带来的额外时间接近于 $O(n)$归并算法消耗的时间。试验证明，本文子串归并算法有效。

5.3 相关研究比较

已有方法实验在机器配置、操作系统、数据等方面各不相同，再现难以与原始效果等同，因此不能简单以速度为评测标准。

本文方法可处理规模大于内存的语料，这是单纯基于后缀数组或后缀树结构查找所不具备的特性。与 Hunt[1] 和 Schurmann[8] 主要基于磁盘建树的方法不同，本文查找过程 IO 操作较少，效率较高。

另外，与 Chen[9] 和龚[10] 算法具有相似点，都以后缀首字符为划分依据将语料划分为多个集合。不同的是，Chen 对后缀进行排序后建后缀树，从树中查找频繁模式。本文算法减少了建树这一步骤，直接对排序后的后缀进行处理，减少了复杂度，从而缩短了处理时间。龚的方法则是对后缀建立倒排索引，通过倒排索引查找频繁模式。在查找时，将相同首字符的后缀全部导入到内存再处理。本文方法则在逐条扫描的过程中自动查找，以较小的内存获取较高的效率。若语料规模进一步增大，龚的方法需继续划分才能查找。而本文方法受语料规模影响较小，可处理远大于内存的语料，语料增大时不

（下转第 169 页）

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