基于逐层剪枝的中文高频重复模式快速提取算法

张海军[1,2] 刘战东[2] 木妮娜[2]
(新疆师范大学初等教育学院 乌鲁木齐 830054)[1]
(新疆师范大学计算机科学技术学院 乌鲁木齐 830054)[2]

摘 要 为了从大规模语料中快速提取高频重复模式,以递增n-gram模型为基础,使用散列数据结构提取重复串,并提出了一种基于低频字符和层次剪枝的逐层剪枝算法,用于过滤低频垃圾字串,减少I/O读写次数。在此基础上,应用改进的字串排序算法,使字符串排序可在$O(n)$时间内完成,从而有效提高重复模式的提取效率。实验表明,该算法是一种有效的重复模式提取算法,其I/O读写次数同语料规模呈线性关系,远小于使用首字符进行语料划分的方法,能快速有效地从规模远大于内存容量的文本语料中提取重复模式,特别适合于大规模语料的高频重复模式提取,对以重复模式为基础的新词识别、术语抽取等具有重要的支撑作用。

关键词 重复串,散列表,低频字串,逐层剪枝,新词识别
中图法分类号 TP391 **文献标识码** A

Rapid Algorithm of Chinese High-frequency Repeat Extraction Based on Hierarchical Pruning

ZHANG Hai-jun[1,2] LIU Zhan-dong[2] Munina[2]
(School of Elementary Education, Xinjiang Normal University, Urumqi 830054, China)[1]
(School of Computer Science and Technology, Xinjiang Normal University, Urumqi 830054, China)[2]

Abstract To extract high-frequency repeats from large-scale corpus, by using the hash table structure, this paper put forward a hierarchical pruning algorithm based on low-frequency character filtration and Cascade Pruning to filtrate low-frequency strings and to reduce the times of I/O reading & writing. On this basis, this paper employed the improved string sort algorithm, which can implement string sort in $O(n)$ time complexity, to improve the efficiency of repeat extraction. According to the experimental data, it can be concluded that this algorithm is an effective method of repeat extraction, in which the relationship between the times of I/O reading & writing and the scale of corpus is linear. The algorithm can effectively extract repeats from text corpus whose scale is much larger than that of the computer memory and can better suport the repeat-based applications such as new words identification, term extraction, etc.

Keywords Repeat, Hash table, Low-frequency strings, Hierarchical pruning, New words identification

1 引言

随着时代进步和互联网的迅猛发展,中文新词和短语在不断大量涌现,比如:“非典”、“超女”、“快男”等等。这些词或短语的主要特征是出现范围广,出现频率高,以重复串的形式出现,如能在大规模语料中快速获取这些频繁出现的重复模式(重复串),就可以快速地发现这些新词和短语,为自然语言处理提供更多方便。高频重复模式的查找方法对中文新词识别、信息检索和术语抽取等技术的发展都是十分重要的[1-5]。

高频重复模式查找的主要困难在于重复串统计,大量的时间和内存空间都耗费在低频模式的统计计算上,效率低,内存占用大,难以满足重复模式快速获取的要求。

本文提出了一种实用的快速重复串查找算法,即通过对文本两次扫描,将内存的一次大量使用转变为多次少量使用,从而有效地减少内存用量。先扫描取得频率低于阈值的字符,如果字符频率低于阈值,那么包含该字符的字串的出现频率必定不会高于阈值;后续扫描实际上是依次提取各种长度重复模式的过程。为了减少扫描过程产生的垃圾串,本文采用级联逐层剪枝的递进扫描方式,来降低内存用量,提高重复模式的提取速度。

2 相关研究工作

目前已经有许多重复模式提取算法,比较典型的有Sequitur算法[5]、基于后缀索引的算法[6-9]和基于递增n-gram模型算法[10]。

Sequitur算法的主要思想是使用产生式规则来表示语料中的重复结构,形成语法规则库,然后每读入一个字符都需对当前的规则库进行调整。规则库中的规则必须满足两个属

到稿日期:2013-07-18 返修日期:2013-10-27 本文受国家自然科学基金项目(61163045,61263044),新疆维吾尔自治区高校科研基金(XJEDU2012S29),新疆师范大学重点学科招标课题(12XSXZ0601)资助。
张海军(1973—),博士,副教授,硕士生导师,主要研究方向为自然语言处理、信息抽取技术;刘战东(1982—),硕士,讲师,主要研究方向为模式识别、图像处理;木妮娜(1963—),博士,副教授,硕士生导师,主要研究方向为模式识别、信息抽取技术。

性：(1)任何由相邻符号组成的双字串在语法规则库中只能出现一次；(2)任何规则被使用的次数必须超过一次。Sequitur算法的处理过程是从起点开始，每次读入一个字符，加到S规则的右部最后，然后调整规则库，使规则库里所有的规则符合上述两个属性。该算法的时间和空间复杂度都是$O(n)$，可用于快速发现语料结构的层次关系，存在的问题是会遗漏某些重复模式，正是这个问题影响了该算法在重复模式查找方面的应用。

基于后缀索引的算法主要有后缀树和后缀数组。令$T=t_1t_2\cdots t_n$表示文本语料，$C_i(T)=T[i\cdots n]$表示T中从i开始的后缀。语料$T=t_1t_2\cdots t_n$共有n个非空后缀：$C_1,C_2\cdots C_n$。由这些非空后缀构成文本语料T的后缀树，通过遍历后缀树寻找最长公共子串可获得重复串。但后缀树的构造时间与字符集的大小有关，不是线性关系；而且建树所需空间是$O(n\log|\Sigma|)$，对处理汉语这样的大字符集语料效率不高。有人提出用后缀数组代替后缀树[6]，目前后缀数组的重复串查找算法的时间和空间的复杂度都可达到$O(n)$，是效果较好的重复串查找算法。基于后缀数组算法的最大问题是，在处理时需要将语料一次性装入内存，否则就不能实现正确查找。但语料规模一般都会远大于内存容量，导致该方法失效。

朴素的递增n-gram模型重复串查找算法，是在整个语料范围内，依次查找2字、3字以及n字长的重复串的过程。该算法思路简单，实现容易，但最大的问题是效率低，时间复杂度是$O(n^2)$，空间开销在最坏情况下也是$O(n^2)$，因此该算法也难以处理大规模语料。

根据以上论述，这3种算法都不能有效处理规模超过内存容量的语料，为此，研究人员设计了很多语料分割方法，试图解决这个难题。

Hunt[8]提出了一种可行的划分方法，该算法对语料进行多次扫描，每次扫描将以某个前缀开始的所有后缀加到后缀树。Hunt算法建树时间复杂度为$\Theta(n^2)$，其所需空间大约为$65n$。Clifford等[11]在Hunt算法的基础上提出了分布式后缀树(disturbed suffix tree)的构建方法，即把语料拷贝到不同结点，每个结点使用Hunt算法独立构建以某些前缀开始的后缀树。Schurmann等[12]在Hunt算法基础上提出Clustered算法，Clustered算法与Hunt算法的区别在于Clustered算法使用散列函数直接定位子树位置。Clustered算法速度比Hunt算法快，但要求可用内存大小至少为语料规模的数倍以上。Chen首先顺序扫描语料，生成所有后缀并写入到临时文件，对临时文件各后缀排序使得具有相同首字符的后缀在同一组，最后对各组后缀分别建立后缀树。Tian等[13]提出的PWOTD (partition and write only top down)算法采用倒排索引保存各字符在语料的所有出现位置，当要生成以某个字符开始的所有后缀构成的后缀树时，根据倒排索引记录的字符出现位置取出所有后缀，并依次加入到后缀树。

龚才春[4]对Tian的PWOTD算法进行了改进，针对大规模语料的所有后缀，按照其首字符划分为若干组，对各组后缀独立建立后缀树，最后将各组结果组合即可得到语料的全部重复模式。对语料后缀的划分使用了倒排索引的方式，以字符为索引词，若字符在位置P出现，则在字符的Postlist中增加P位置对应的后缀。与PWOTD不同的是，在Postlist中直接保存后缀而不是字符的出现位置，能有效减少处理每一个索引词时都需要将语料分批导入到内存所带来的巨大I/O开销。

以上两种算法虽然可以有效地减少内存用量，但恰恰是为了节省内存，这两种算法需要频繁地进行I/O访问。对于Tian的方法，由于Postlist中存储的是字符位置索引，需要频繁的I/O访问来实现语料中字符的载入；对于龚才春的方法，因需要根据起始字符分别进行语料划分，I/O访问的次数至少是常用字符的数量，如果语料再大些，当内存无法容纳以单个字符为根结点开头的后缀树时，I/O访问的次数将会以指数递增，导致该算法难以处理大规模语料。

为解决前述矛盾，通过深入研究中文重复模式性质，本文提出一种可靠的重复串查找方法，具体思想如下：首先进行一次语料扫描，找到出现频率低于设定阈值的字符，将这些字符作为语料剪枝字符，其作用是减少低频垃圾串的产生；通过研究长串与短串之间的关系，对朴素的递增n-gram算法进行改进，以散列结构为基础，分层次逐步提取不同长度的重复串；在重复串提取时，用短串模式集合过滤相邻长串模式，以减少低频垃圾串的出现；使用一种改进的字符串排序方法实施字串排序，最后应用外部排序实现重复模式提取。

本文的主要贡献包括：(1)提出了一种可靠的低频串剪枝方法，即根据事先设定的阈值，动态提取分割字符，从开始阶段就有效地控制低频垃圾串的产生；(2)提出了一种分层级联的重复模式查找算法，该算法能有效地控制内存的用量，在有限内存范围内，通过有限次的I/O访问完成大语料乃至超大语料的重复模式提取。

3 基于逐层剪枝的重复模式提取算法

3.1 重复模式提取的形式化描述

假设Σ是有限的字符集合，S是由Σ中的字符所构成的有限的字符序列，$S=c_1c_2c_3\cdots c_n$，S的长度为n。$S[i]$表示S中的一个字符，其中$1\leqslant i\leqslant n$，$S[i,j]$表示序列S中的一个字符串。$C=S_1\#S_2\#S_3\cdots\#S_m$表示由$m$个文本构成的字符语料，其中#是$\Sigma$集合中的一个字符，表示文本结束，各种标点符号如“。”、“?”、“!”等都可作为文本结束符。设$R=c_1c_2\cdots c_k$并且R中不包含文本结束符#，如果在语料中至少存在两个位置p_1和p_2，使得$S[p_1\cdots p_1+k-1]=S[p_2\cdots p_2+k-1]=R$，则称$R$为语料$C$中的重复模式，如果$R$出现的频率高于预先定义的阈值$\lambda$，则称$R$为高频重复模式。重复模式查找就是在语料$C$中寻找各种长度的重复模式$R$的过程。

3.2 基于低频字符的低频模式过滤方法

在递增n-gram模型中，如果将文本语料不加处理直接进行高频模式查找，必然会存在大量频率低于阈值λ的垃圾字串，造成内存浪费。由重复模式的性质可知，对于模式$R=c_1c_2\cdots c_k$，若$f(R)\geqslant\lambda$，必然存在$f(c_1)\geqslant\lambda,f(c_2)\geqslant\lambda,\cdots,f(c_k)\geqslant\lambda$。上述命题的逆否命题是，如果$f(c_i)<\lambda,c_i\in\Sigma$，则包含字符$c_i$的任意模式$R$，一定会有$f(R)<\lambda$成立。根据这个理论，可以通过一次语料扫描，取得满足$f(c_i)<\lambda,c_i\in\Sigma$条件的所有字符$c_i$，将包含$c_i$的集合$\sum_0$作为剪枝字符集合，在重复串扫描中，对于模式$R$中的任意字符$c_x$，如果有$c_x\in\sum_0$，则可将模式$R$滤掉，从而在字符层面上实现低频模式的

过滤，有效地节省内存。

3.3 基于层次剪枝的低频模式过滤方法

依据前面的形式化描述，通过研究发现：重复模式 $S[p_1\cdots p_1+k-1]=S[p_2\cdots p_2+k-1]=R$ 的长度为 k，如果 R 的频率为 m，模式 $X=c_iR$ 或 $X=Rc_j$（其中 $c_i\in\Sigma,c_j\in\Sigma$）的长度为 $k+1$，设其出现频率为 m'，根据重复模式的性质可知，$m'\leqslant m$，也就是说，一个重复模式的出现频率应该小于或等于其子模式的出现频率。进一步地，如果 R 的出现频率小于阈值 λ，即 $m<\lambda$，那么 X 的出现频率一定小于阈值 λ，即 $m'<\lambda$。可见，如果其子串 R 是频率低于设定阈值的垃圾串，那么该长串 X 也一定是频率低于阈值的垃圾串。根据这个性质，可以考虑使用低频垃圾模式集合，对候选模式进行过滤，实现低频字串剪枝。

设长度为 k 的字串模式的全集为 Ω，频率低于阈值 λ 的垃圾串集合为 $\bar{\varphi}$，出现频率大于或等于阈值 λ 的重复模式集合为 $\varphi=\Omega-\bar{\varphi}$。因对 $k\geqslant 2$ 时有 $|\varphi|\ll|\bar{\varphi}|$，所以可考虑用 φ 代替 $\bar{\varphi}$ 来实现低频垃圾串的过滤，以节省内存，提高检索速度。因 $R\in\Omega$ 且 $\varphi\cap\bar{\varphi}=\phi$，有（$R\in\varphi$ 且 $R\notin\bar{\varphi}$）或（$R\notin\varphi$ 且 $R\in\bar{\varphi}$）成立。所以，根据这种互斥关系，使用规模较小的 φ 作为过滤集合即可。过滤原则是，对于模式 $X=c_iR_1$ 或 $X=R_2c_j$，当 $R_1\in\varphi$ 且 $R_2\in\varphi$ 时，X 可作为候选模式；否则，将 X 作为垃圾串抛弃。

3.4 基于外部排序的重复模式归并

为了处理容量远大于内存的大规模语料，需要对语料进行分割，以便在内存范围内快速查找重复串。本文方法只需将语料随机地划分为规模不超过预定大小的组块，然后按照组块依次地进行重复模式提取。由于任意相邻的两个或多个汉字字符都有组合的可能性，重复模式提取时会产生大量的频率低于阈值的垃圾模式，通过前面两个小节论述的垃圾字串剪枝方法，可以有效地减少垃圾模式，构造候选重复模式集合。

为减少内存用量，每个组块所产生的候选重复模式都会被保存在临时文件中，但这些文件中的候选模式并不是整个语料中满足出现频率阈值的最终重复模式集合，还需对这些候选模式进行归并处理。为了处理大规模语料，本文采用外部排序方式来合并重复模式。

基于递增 n-gram 模型提取重复模式，每次只用固定长度的窗口来扫描整个语料，根据这一特点，本文提出使用改进的基数排序（radix sort）来提高内部排序效率[15]。基数排序虽有较高的处理效率，但一般适用于位数相同的数值型数据排序[14]。若要实现对字符串的基数排序，需预先对字串进行处理。为此，考虑用确定的整数编号来表示汉字字符，从而可将重复模式字串转化为与之具有相同长度的整型数组。因汉字与数值编号一一对应，所以字串与整型数组之间必然是一一映射关系。根据这种对应关系，如果实现了对整型数组的排序，也就实现了对字串的排序功能，该排序算法的时间复杂度为 $O(n)$。

3.5 算法描述

根据以上算法思路，基于散列（Hashtable）结构和递增 n-gram 模型，设计了大规模语料的重复模式查找算法，具体描述如图 1 所示。

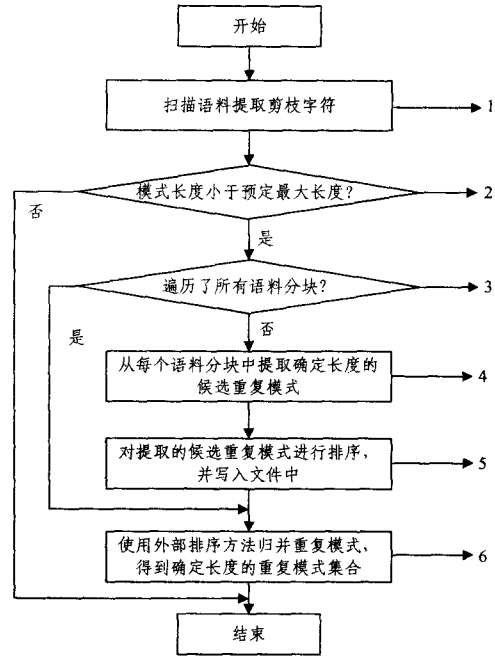

图 1 基于逐层剪枝的重复模式提取算法

根据图 1，模块 1 用于扫描整个语料，提取出现频率低于阈值的字符构成剪枝字符集合；模块 2 控制循环，长度从 2 开始，进行递增长度重复模式提取；后续模块描述了如何在大规模语料中提取重复模式，模块 3 进行的是对整个语料进行遍历；模块 4 使用层次剪枝和低频字符剪枝方法对每个语料分块提取确定长度的候选重复模式；模块 5 对提取得到的候选重复模式进行排序并写入文件中；当语料中的所有分块处理完成后，模块 6 使用外部排序方法对每个块中提取的候选重复模式的出现频率进行归并，最终得到某一长度的重复模式集合；然后重复上述过程，进行下一长度重复模式的提取。

3.6 算法的时间消耗分析

整个算法的时间消耗由 3 部分组成，第 1 部分是模块 1 实现的低频剪枝字符集合提取；第 2 部分由模块 3 控制的长度递增逐层剪枝的循环结构构成，其内部实现基于内存的候选重复模式快速查找，结果集合排序后写入临时文件；第 3 部分由模块 6 组成，用于实现对整个语料产生的所有临时文件外部归并，从而取得频率高于阈值的全部重复模式。

第 1 部分中，散列结构检索和插入的时间复杂度是 $O(1)$，设整个文本语料的总长度为 n，该部分的时间消耗为 $n\times O(1)+fR(n)=O(n)+fR(n)$，其中 $fR(n)$ 为读整个语料所需时间。

第 2 部分中，确定长度的重复模式查找从模块 4 开始，在其范围内包括赋值、判断、重复模式的散列检索和插入操作，该部分的复杂度为语料规模 $O(n)$；模块 5 是对每个语料块所产生的候选重复模式进行排序，设当前长度为 k 的候选模式数量为 M，存在 $M\leqslant n-k+1$，由于使用低频字符过滤和逐层剪枝过滤低频垃圾串，必然存在 $M\ll n$，$O(M)\ll O(n)$，因此排序时间复杂度的上限为 $O(n)$，针对每个长度的候选模式提取和排序所需时间为 $O(n)+O(n)$；该部分还需多次遍历读取整个语料，并将排序后的候选重复模式写入临时文件，前者与语料规模有关，后者应与候选重复模式的规模有关，因需要提取长度从 2 到 k 的重复模式，提取的次数为 $k-1$，所以这部分总时间消耗为 $(O(n)+O(n))\times(k-1)+fR(n)\times(k-1)$

$+\gamma\times fW(n)=O(n)+(k-1)\times fR(n)+\gamma\times fW(n)$，$fW(n)$为写出整个语料规模字节所需的时间，$\gamma$须由试验确定。

第3部分是对遍历语料获得的多个重复模式集合进行外部归并，需要大量的文件读写操作，时间消耗同语料规模具有某种线性关系，假设其为：$\alpha\times fR(n)+\beta\times fW(n)$，具体参数需根据试验确定。

综上，对整个语料进行重复模式提取的时间为：

$$\begin{aligned}&O(n)+fR(n)+O(n)+(k-1)\times fR(n)+\gamma\times fW(n)+\\&\quad\alpha\times fR(n)+\beta\times fW(n)\\&=O(n)+K\times fR(n)+\alpha\times fR(n)+(\gamma+\beta)\times fW(n)\\&=O(n)+K\times fR(n)+\alpha\times fR(n)+\beta'\times fW(n)\end{aligned}$$

对于规模远大于内存的大语料，整个重复模式提取过程中I/O读写操作占有很大的比重，由于I/O读写操作速度要远慢于内存操作速度，因此I/O对语料的读写次数就决定了重复模式的提取效率。

4 实验及结果分析

4.1 实验条件

根据前述算法，使用Microsoft C# 2008进行了算法实现。实验的计算环境是windows XP SP3，2G内存，所用语料是搜狐试验室提供的大规模网络语料，该语料需经解析以便将压缩在一起的网页文件转化成纯文本语料；实验中对大约400G的Sogou网页语料进行解析，提取了纯文本语料16G，最大词长$MaxLen=10$。

4.2 实验结果

为了验证该方法的正确性，使用了10kbytes的纯文本语料，采用手工标注的方法标注所有频率阈值$\lambda=2$、字串长度从2到10的重复模式，本文方法的准确率和召回率皆为100%。

使用10组1Mbytes的纯文本语料，采用递增的n-gram重复模式提取方法作为基准，针对频率阈值$\lambda=2$字串长度从2到10的重复模式，结果本文方法所有重复模式提取的准确率和召回率皆为100%。从实验的角度来讲，本文方法能准确可靠地从语料中提取重复模式。

针对不同规模的语料，处理频率阈值为10，模式长度从2到10，本文算法的处理速度如表1所列。

表1 算法处理速度表

语料容量	处理时间(ms)	处理速度(M/s)
2 G	3351594	0.611052532
4 G	6729327	0.608678911
6 G	10266248	0.598465908
8 G	13830475	0.592315144

从表1中可见，随着语料规模的增长，重复模式提取速度有所降低，但基本保持在0.59M/s左右；本文方法特别适用于大规模语料的高频重复模式提取，当频率阈值进一步增大时，处理速度会得到更为显著的提升。

4.3 实验数据分析

为了验证算法的剪枝性能，针对不同规模的语料进行了对比试验(频率阈值$\lambda=100$)，结果详见表2。

表2 I/O操作实验数据表

语料容量(G)	IO读取字节数	IO写入字节数	读取语料次数[1)]	写出次数[2)]
1	12256691564	1612032136	11.4	1.5
4	56013080572	12822437898	13.0	3.0
6	87476205463	22809723148	13.6	3.5
8	119697130280	33776558000	13.9	3.9
12	187937434348	58817845729	14.6	4.6
16	258507654500	86408517723	15.0	5.0

从表2中可看出，随着语料规模的增长，I/O读写语料的次数在增加，但增长的速度逐渐趋缓；而且，对不同规模的语料来讲，读取与写出的次数相互关联，二者的差值是最大词长(实验中为10)，这是因为提取全部重复模式的操作需对语料进行最大词长次数的扫描，多余的读取次数恰恰是由于读取了操作中写出的候选重复模式数据造成的。从图2中可见，相同容量语料的读写次数之差保持定值，也可证明上述结论。

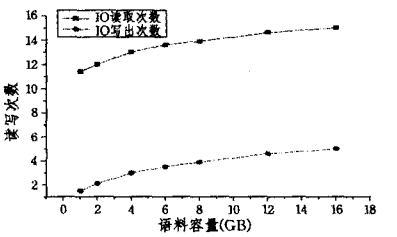

图2 递增语料容量I/O读写次数图

同前面介绍的方法相比，本文方法对语料的读写次数要小很多。因为前述的方法根据首字符划分语料，当语料为内存规模时，对语料的读写次数大约为字符的类型数(字符集的规模)，当语料更大时，需对语料进行多级首字符划分，导致对语料的I/O读写次数呈指数增长。

为了分析逐层剪枝(HP)对读写次数的影响，对比不用HP过滤方法，进行了多组相同语料的平行实验，结果详见表3。从表中可见，对相同容量的语料，使用HP过滤与不用过滤相比，读写数据量都大幅度减少。根据前面分析，读写操作相互关联，所以只讨论写出数据量即可(当然讨论读取数据量也会得出相同结论)。为此，本文定义了过滤比来标识HP过滤效果，定义如下：

$$\text{过滤比}=\frac{(\text{未过滤写出字节数}-\text{HP过滤写出字节数})}{\text{未过滤写出字节数}}\times 100\%$$

表3 平行对比实验结果

语料容量	方法对比	IO读取字节数	IO写出字节数	过滤比
1G	未过滤	30597850261	19945043593	91.9%
	HP过滤	12264013408	1612032424	
2G	未过滤	69105021161	47532297464	90.1%
	HP过滤	26284318853	4712245955	
4G	未过滤	149180690491	105981792558	87.9%
	HP过滤	56020756684	12822498097	
8G	未过滤	320093118126	234164934675	85.6%
	HP过滤	119697130280	33776558000	

根据表3中的数据，使用HP进行低频字串过滤，能将85%以上的垃圾字串滤去；可见HP具有非常明显的过滤效果，能有效减少I/O的读写次数，提高重复模式查找效率。但随着语料规模的增长，过滤比会逐渐降低，这是因为随着语料

1) 读写的总字节数是语料容量的倍数

2) 写出的总字节数是语料容量的倍数

规模的增长，字符之间的组合变得更加复杂，造成越来越多垃圾字串通过 HP 过滤，进入到候选重复模式集合中，导致过滤效果有所降低。当语料容量增长到特定规模后，过滤比会保持一个常量，但这个特定规模在实际应用中难以达到。从实验结果来看，语料规模达到特定规模之前，HP 的过滤效果是非常显著的。

4.4 同其它方法比较

目前在处理语料规模大于内存容量的重复模式提取方法中，文献[4]所提出的是比较有代表性的方法，但因实验条件和实验语料不具有可比性，本文没有进行量化比较分析。

根据前述定性分析，因 I/O 读写的速度要远低于内存处理速度，在语料规模超过内存容量后，对于 I/O 的操作次数就成为衡量算法处理性能的关键指标。如在处理中文语料时，文献[4]中方法在语料规模增大到内存容量时其 I/O 操作次数约为汉字字符集规模，不超过 7000，当语料规模进一步增大时，其需要以第二字符进行二次划分，导致 I/O 操作次数呈指数级增长，会严重影响其处理效率；而本文方法的 I/O 操作次数同语料规模是一种线性关系，因此对语料的规模不敏感。前述实验中当处理规模为 32GB 的中文语料时，其 I/O 操作次数约为 16.3+6.3=22.6 次。

当然，文献[4]中方法适用于并行计算，若在并行环境中其处理效率会非常高，而本文算法因需要逐层剪枝和全局垃圾字串过滤，难以用于并行环境中。

结束语 使用逐层剪枝的低频垃圾串过滤，可有效地节省内存消耗，极大地减少 I/O 读写次数，提高重复模式的提取效率。实验表明，本文算法能极大限度地过滤低频字符串，快速地进行大规模语料的重复模式提取，特别适合于在大规模语料中提取所有高频重复模式。

通过大量研究，我们大胆预测：虽然汉字组合数量从理论上讲是无穷的，但当语料规模增大到特定程度后，重复模式的总量基本保持常量。这是因为，汉字字符组合必须要遵循汉语语言习惯，导致重复模式的数量是有限的；当然，一般情况下，我们所寻找的重复模式集合只是这个集合的子集。

尽管使用逐层剪枝过滤方法提高了重复模式提取效率，但低频垃圾模式的过滤效果尚有很大的改进空间；基于语料块的局部重复模式归并算法还需改进，以进一步减少 I/O 读写次数；如有条件，可对更大规模的语料进行研究和处理，以期发现实验中尚未发现的规律和趋势，这些都是本研究下一步需要处理的问题。

参 考 文 献

[1] 黄昌宁，赵海. 中文分词十年回顾[J]. 中文信息学报，2007，21(3)：8-19

[2] 邹纲，刘洋，刘群，等. 面向 Internet 的中文新词语检测[J]. 中文信息学报，2004，18(6)：1-9

[3] 张海军，史树敏，朱朝勇，等. 中文新词识别技术综述[J]. 计算机科学，2010，37(3)：6-10

[4] 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

[5] Nevill-Manning C G，Witten I H. Identifying Hierarchical Structure in Sequences：A Linear-Time Algorithm[J]. Journal of Artificial Intelligence Research，1997，7(1)：67-82

[6] 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

[7] Larsson N J，Sadakane K. Faster Suffix Sorting[D]. Lund，Sweden：Department of Computer Science，Lund University，1999

[8] Martin F-C，Paolo F，S M. On The Sorting Complexity of Suffix-Tree Construction[J]. Journal of ACM，2000，47(6)：987-1011

[9] Anisa A H，Maxime C，Lucian I，et al. A comparison of index-based lempel-Ziv LZ77 factorization algorithms[J]. ACM COMPUTING SerVey，2012，45(1)：5

[10] 郑家恒，李文花. 基于构词法的网络新词自动识别初探[J]. 山西大学学报：自然科学版，2002，25(2)：115-119

[11] Clifford R，Sergot M. Distributed and Paged Suffix-Trees for Large Genetic Databases[C]// Proceedings of 14th Annual Symposium on Combinatorial Pattern Matching. 2003：70-82

[12] Schurmann K-B，Stoye J. Suffix Tree Construction and Storage with Limited Main Memory[D]. Bielefeld，Germany：University of Bielefeld，2003

[13] Tian Y，Tata S，Hankins R A，et al. Practical Methods for Constructing Suffix Trees[J]. The VLDB Journal，2005，14(3)：281-299

[14] Cormen T H，Leiserson C E，Rivest R L，et al. In Introduction to Algorithms (2nd Ed) [M]. Cambridge，MA：MIT Press，2001

[15] 张海军，潘伟民，木妮娜，等. 一种自定义顺序的字符串排序算法[J]. 小型微型计算机系统，2012，33(9)：1968-1971

(上接第 242 页)

[8] Schölkopf B，Smola A. Learning with Kernels[M]. Cambridge：MIT Press，2002

[9] Cristianini N，Shawe-Taylor J，Elisseeff A，et al. On kernel-target alignment[C]//Proceedings of Advances in Neural Information Processing Systems. 2001：367-373

[10] Wang Wen-jian J，Xu Zong-ben，Lu Wei-zhen，et al. Determination of the spread parameter in the Gaussian kernel for classification and regression[J]. Neurocomputing，2003，55(10)：643-663

[11] Paclík P，Novovič ová J，Pudil P，et al. Road sign classification using Laplace kernel classifier[J]. Pattern Recognition Letters，2000，21(13/14)：1165-1173

[12] Fleuret F，Sahbi H. Scale-invariance of support vector machines based on the triangular kernel[C]//Proceedings of 3rd International Workshop on Statistical and Computational Theories of Vision. Nice，2003