计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(8):262−267 [doi: 10.15888/j.cnki.csa.007052]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于字符树结构的高性能中文词库技术①

杨光豹[1], 杨丰赫[2], 郑慧锦[3]

[1](浙江广播电视大学 青田学院, 青田 323900)
[2](东南大学 网络空间安全学院, 南京 211189)
[3](浙江青田县职业技术学校, 青田 323900)

摘　要: 海量中文信息处理是大数据处理的一个分支, 而利用大数据技术进行中文信息处理一定离不开中文分词, 所以中文分词技术是大数据中文信息处理的基础性技术. 中文分词技术自本世纪以来, 一直在性能与精确度两个方向在推进; 在性能方面主要以改进分词扫瞄算法, 改进词库存储技术与查询方式来提高性能. 在精确度上主要是对未登录词与歧义词的甄别与处理方法进行改进. 本文摒弃了通过词库索引查询的思想, 提出一种基于字符树的词库存储结构. 它的分词速度是普通折半法的 35 倍, 占用内存只是它的 1/5. 它将为大数据技术在处理中文信息时在性能上推进了一大步.

关键词: 字符树; 中文分词; 散列法; 折半法; 时间复杂度

引用格式: 杨光豹,杨丰赫,郑慧锦.基于字符树结构的高性能中文词库技术.计算机系统应用,2019,28(8):262–267. http://www.c-s-a.org.cn/1003-3254/7052.html

High Performance Chinese Lexicon Technology Based on Character Tree Structure

YANG Guang-Bao[1], YANG Feng-He[2], ZHENG Hui-Jin[3]

[1](Qingtian College, Zhejiang Radio &TV University, Qingtian 323900, China)
[2](School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China)
[3](Zhejiang Qingtian Vocational and Technical School, Qingtian 323900, China)

Abstract: Massive Chinese information processing is a branch of big data processing, and the use of big data technology for Chinese information processing must be inseparable from Chinese word segmentation, so Chinese word segmentation technology is the basic technology of big data Chinese information processing. Chinese word segmentation technology has been advancing in performance and accuracy since this century. In terms of performance, it mainly improves the segmentation scanning algorithm, the word bank storage technology, and query method to improve the performance. In terms of accuracy, it is mainly to improve the processing method of unregistered words and ambiguous words. This paper gives up the idea of searching by lexicon index and proposes a lexicon storage structure based on character tree. Its segmenting speed is 35 times faster than the normal half method, occupying only 1/5 of its memory. It will be a big step forward in the performance of big data technology in processing Chinese information.

Key words: character tree; Chinese word segmentation; hash; binary query; time complexity

引言

利用大数据技术处理海量中文信息是当今很成熟的技术, 而中文分词作为中文信息处理的基础, 它是大数据处理中文信息的基础. 而中文分词的性能与准确度一直是相互制约的. 业内对该技术的研究一直在性能与精确度两个方向在推进. 中文分词方法的研究从

① 收稿时间: 2019-02-22; 修改时间: 2019-03-22; 采用时间: 2019-03-25; csa 在线出版时间: 2019-08-08

大的方向来讲, 大体分为两大类: (1) 基于词库的字典的机械分词法. (2) 采用统计学理论的统计分词法. 比如文献[1]是使用人工制作的分词语料进行特征信息学习的机器学习分词法. 文献[2,3]是通过双向 LSTM 神经网络模型进行分词. 文献[4]利用上下文词长特征作为分词特征, 从上下文信息中获取信息进行分词,这些都属于统计分词法. 这类分词法在对未登录词与歧义词的处理上占优势, 但性能上不及机械法. 一种好的分词技术往往都是采用综合方法, 以机械分词为基础, 再融入统计法以提高分词精确度. 例如文献[5]利用统计学上的概率, 引入层次分析法将多种切分法优劣排序, 帮助选择最优排序法, 它在一定程序上提高了准确率. 在机械分词中, 为了提高性能, 文献[6]提出了前四字 hash 索引树技术. 文献[7]提出了首字 hash 查询的方法, 文献[8,9]提出了双字 hash 法; 文献[10]提出了对四字 hash 索引的 trie 树进行改进, 文献[11]提出了利用首字 hash 法进行正向与逆向的多方案分词的选择以减少分词的歧义. 而文献[12]却提出了首字拼音首字母表. 目前为止, 基于词库的中文分词都有一个共同特点就是词库结构都是: 索引+余词表. 在索引上采用各自认为速度快, 占用内存小的方式来进行查询. 目前业内普遍认为采用前 4 字 trie 索引树法的分词技术性能最好. 而本文在实际编程中发现, 利用字符树结构的分词技术能更好地进行中文分词, 无论从时间复杂度还是空间复杂度, 字符树分词技术都更胜于其它各种词库技术, 它能进一步改善大数据处理中文信息的性能.

1 字符树分词原理

1.1 单字符树原理

单字符树是在各个节点上都保存单个字符的树. 将词库文件加载到内存时, 所有的词都将先被分割成一个个的单字, 然后逐个“挂接”在一棵树上. 一个词的所有字加载完后在最后一个字节点上作词串标记, 然后加载第二个词的各个字, 如果这棵字符树上该位置已经存在要加载的字, 则视为已加载 (不覆盖), 但如果是词的最后一个字符, 则要对它的词串标记进行修改. 整棵树的构造如图 1 所示.

单字符树的各个节点都采用同样的数据结构, 每个节点中都有一个词串标记 isword, 如果其值为 true 则表示从根到该节点为止的路径构成一个词, 否则仅仅是词的中间节点. 其数据结构如图 2 所示.

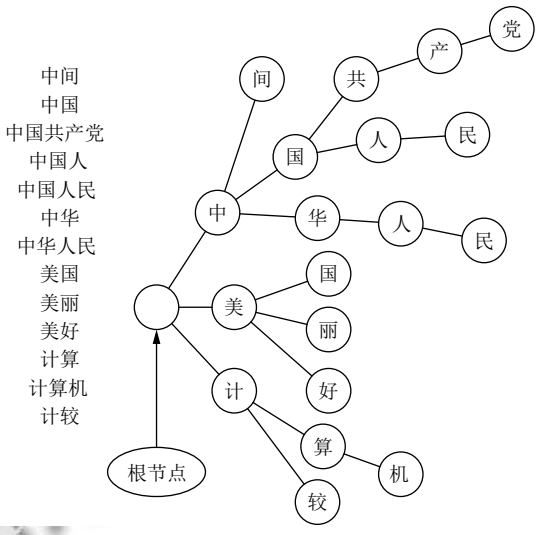

图 1 单字符树词库构造举例

```
struct Node {
WCHAR code=L'\0'; //字符内码
bool isword=false; //词串标志
int size=0; //子节点数量
int datas=0;
Node *next; //指向子节点数组的首地址
}
```

图 2 单字符树词库节点数据结构

查询一个词只需要从根节点开始, 例如查询“中国”这个词, 只需要从根节点开始先找“中”, 找到后再在“中”的子节点中找“国”找到后再看这个“国”的节点中的词串标记是否是 true,如果是 true,则说明词库中存在“中国”. 否则就不存在. 这样构造起来的字符树最大的好处是不会出现无效查询, 当查询词库中存在的词, 比如想查询“千方百计”, 它从树根逐级查询会有一条路径存在, 最后依据路径末端节点的词串标记来判断. 查询词库中不存在的字符串时, 要么路径不存在, 要么路径末端节点的标记是 false.

1.2 多字符树原理

多字符树是从单字符树演变过来的一种变异树. 它的设计思路与单字符树类同. 只是为了减少节点数, 把字符树中出度为 1, 词串标记为 false 的节点字符连在一起放在一个节点上, 直到节点出度大于 1 或标记位为 true 为止, 但它的查询思路与单字符树是一样的, 都是从树根开始向各个分支逐级查询, 加载时相同的前缀也只加载一份.

多字符树的构造过程是这样的: 例如“ABCDEF”是一个词, 开始加载这个词时, 将它作为整体挂在根节

点的一级子节点上, 此时如果加载第二个词“ABCMN”, 由于有共同的前缀“ABC”, 于是将原来的节点“ABCDEF”进行“裂变”, 产生共同前缀的节点为父节点“ABC”, 子节点为“DEF”, 然后再把要插入的“ABCMN”这个词的共同前缀去掉, 剩下的字符构成一个节点“MN”,并把它挂在“ABC”节点下的子节点数组中. 此时如果要加载 ABC 这个词, 则只需要将“ABC”节点的标记位设为 true 即可. 而没有共同前缀的词都挂在根节点的一级子节点上, 依此类推将所有词库文件中的词加载到这棵树上. 整棵树的构造如图 3 所示, 节点数据结构如图 4 所示.

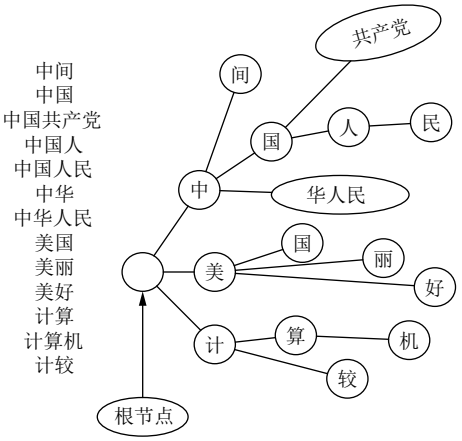

图 3 多字符树词库构造举例

```
struct  ClusterNode{
WCHAR code; //节点首字内码
bool isword; //词串标记
short size ; //子节点数组长度
short leaflen; //节点字符数组长度
WCHAR *leaf; //指向字符数组的指针
ClusterNode* next; //指向子节点数组的指针
};
```

图 4 多字符树词库节点数据结构

多字符树结构的词库节点总数比单字符树少, 但它的查询过程与单字符树相同, 需要逐个比较字符, 所以它的查询性能与单字符树是相同的. 它总节点数少是否意味着内存占用少呢? 事实并非如此. 本实验词库加载成多字符树时, 节点总数是 291 893 个, 约是单字符树的节点数目的 3/4; 它平均每个节点大约拥有 1 个词, 但因为多字符树中的每个节点保存的字符数目是不确定的, 所以它不能全部直接存储在节点上, 只能将节点字符串中第一个字符保存在节点上, 而多出的字符只能以数组的形式用指针“挂”在节点上, 这样以来, 每个节点必须要多出一个指针字段进行挂接字符串, 而且多出的字符需另外申请空间来存放, 这将在内存占用上比单字符树要大得多, 后文实验数据会证明这一点.

2 各词库查询性能分析

无论是单字符树还是多字符树, 它们都不是索引树. 因为它们没有所谓的“余词正文”, 所有的词都已经在这棵树上, 它只是把词库的存储方式从“表”变成“树”, 它不需要索引.

为了比较整词法, trie 索引树法以及单字符树与多字符树等在词库成功查询上的性能区别, 我们采用顺序访问的平均访问长度作为衡量各种词库性能的指标, 以此比较它们在查询性能上的优劣.

2.1 字符树词库查询长度统计分析

由于单字符树与多字符树在查询原理上相同, 所以查询性能相同, 在性能比较时以单字符树为代表参与. 之所以引用两种结构的字符树是为了研究它们占用的内存区别.

单字符树词库构造完成后各节点的统计数据如表 1 所示. 本实验词库中最长的词是 16 个字, 所以单字符树的最大层次是 16, 每一层的节点中如果有孩子, 它会派生出下一层的子节点, 它将作为下一层节点的父节点统计, 而没有孩子的节点不作为下一层节点的父节点统计,节点平均访问长度是指从父节点到本节点需要的平均访问次数,词串的访问长度是指一个词各节点的访问长度之和, 而词串平均访问长度是指同一长度的词串的访问长度的平均值. 由表 1 可知, 字符树中各词串的平均访问长度是 3371, 其主要由一级节点数量 (常用汉字数量) 决定. 它的词库查询时间复杂度为 O(1)

2.2 trie 索引树词库查询长度统计分析

trie 索引树结构以索引字数分为四类: 前 1 字 trie 索引树 trie1, 前 2 字 trie 索引树 trie2, 前 3 字 trie 索引树 trie3, 前 4 字 trie 索引树 trie4. 它们前 n 字 (n 为 1, 2, 3, 4) 与字符树结构相同, 但字数长度超过 n 的词, 其剩余部分全部存在同一张“余词表”中, 所以将余词表视为同一层次的节点, 以 trie4 为例, 其平均访问长度统计如表 2 所示. 各类 trie 索引树的平均访问长度统计方法与表 2 相同, 限于篇幅, 在此不再一一举出.

表 1 单字符树词库平均访问长度统计表

节点级别	父节点总数	节点总数	平均节点数	节点平均访问长度	词串平均访问长度
1 字节点	1	6712	6712	3356.5	3356.50
2 字节点	5540	156 155	28.2	14.59	3371.09
3 字节点	69 933	136 411	1.95	1.48	3372.57
4 字节点	62 492	70 639	1.13	1.07	3373.63
5 字节点	12 556	14 186	1.13	1.06	3374.70
6 字节点	7607	7962	1.05	1.02	3375.72
7 字节点	4870	5066	1.04	1.02	3376.74
8 字节点	3064	3149	1.03	1.01	3377.76
9 字节点	2011	2044	1.02	1.01	3378.76
10 字节点	1330	1340	1.01	1.00	3379.77
11 字节点	793	797	1.01	1.00	3380.77
12 字节点	527	530	1.01	1.00	3381.77
13 字节点	275	275	1.00	1.00	3382.77
14 字节点	172	172	1.00	1.00	3383.77
15 字节点	72	72	1.00	1.00	3384.77
16 字节点	23	23	1.00	1.00	3385.77
		加权平均			3371.99

表 2 前四字 trie 树词库平均访问长度统计表

节点级别	父节点总数	节点总数	平均节点数	节点平均访问长度	词串平均访问长度
1 字节点	1	6712	6712	3356.5	3356.50
2 字节点	5540	156 155	28.19	14.59	3371.09
3 字节点	69 933	136 411	1.95	1.48	3372.57
4 字节点	62 492	70 639	1.13	1.07	3373.63
余词	--	15 415	15 415	7708	11 081.6
		加权平均			3797.578

2.3 整词词库查询长度分析

整词词库的平均访问长度直接受词库总词条数量影响, 假设词库总词条数量为 n, 则它在顺序查询时, 成功查询的平均访问长度为 $n/2$, 失败查询长度为 n. 所以它的词库查询的时间复杂度为 O(n). 它的性能会随着词库的增大而变慢. 本实验词库中它的平均访问次数是 139 548 次, 是字符树平均访问次数的 41 倍. 随着词库条目的增大, 两者性能差距将越来越大.

2.4 各类词库查询性能对比

现将本实验词条集构成的字符树词库、各类 trie 索引树词库以及整词词库的平均访问长度与余词数量统计在表 3 中.

由表 3 可知, 平均访问长度是字符树最短, 它的平均访问长度接近于一个常数 (常用汉字数 6712 的一半), 整词词库最长, 它接近于词库总数的一半, 而 trie 索引树则介于两者之间, 对前 n 字的 trie 索引树来说, n 越大, 级数分的越多, 余词数越少, 它的平均访问长度就越短, trie 索引树就会向单字符树演变; 当 n 越小, 它的级数越少, 余词表越大, 它的平均访问长度就越长, 于是它将向整词词库演变. 所以字符树词库与整词词库是 trie 索引树词库的两个极端, 前者词库内查询时间复杂度为 O(1), 它不随词库规模增大而增大; 而后者词库内查询时间复杂度为 O(m), 它随词库的规模 (词条总数为 m) 增大而增大; 而 trie 索引树性能在这两者之间, 当词库增大时, 随着余词数量增大, 其性能就会降低, 尤其在大规模词库中, trie 索引树词库的余词大量存在, 余词表对性能影响将上升为主要矛盾. 而字符树词库的查询词条的时间复杂度为 O(1), 它不受词库总条数影响, 且不存在余词表, 所以在大规模词库中将会突显它的优势.

3 字符树在扫瞄算法上的优势

基于词库的中文分词是建立在“最大匹配算法”的思想上, 要从待分词的字符串中找出尽可能字数多的

词, 分词时提取的词长度越长越好, 分词结果中词的个数越少越好.

表 3 各类词库平均访问长度统计表

词库类型	余词数量	词串平均访问长度
整词词库	--	139548
前 1 字 tire 树	272587	136472
前 2 字 tire 树	153517	45592
前 3 字 tire 树	74838	13405
前 4 字 tire 树	15415	3798
单字符树	0	3372

3.1 整词词库分词扫瞄的缺陷

传统整词法存在大量的无效查询, 效率低下. 比如要找出字符串“ABCDEFGHIJKLM”中最长的词. 如果词库中最长词的长度是 10 个, 则它必须先取长度是 10 的字符串 A-J 与词库中词比较, 查找是否有相同的, 如果有则提取该字符串, 如果没有, 则第二次取长度为 9 的字符串 A-I 来比较, 依此类推, 逐一缩短字符串长度进行查询. 在最坏的情况下, 10 次查询有 9 次无效的, 它的分词时间复杂度为 O(n×m), (n 为待分词总字符数, m 为词库最长字符数).

3.2 Trie 索引树分词扫瞄的缺陷

Trie 索引树能够将词长度小于树深度的词快速找到并确定, 消除了它们的无效查询问题. 所以对于词长度小于树深度的词, 它的时间复杂度与字符树相同. 然而, 对于词长度大于或等于树的深度时, 依然与传统方法一样, 存在无效查询的问题, 比如上例字符串中如果 ABCD 是一个词, 它从首字, 次字, 三字, 四字都找到了, 此时它却无法确定 ABCD 就是最长的词, 因为最长的词有可能是 ABCDE, 也有可能是, ABCDEF…甚至是更长的字符串. 它既然无法确定 ABCD 是最长的词, 就只能逐个去查询对比才能确定最长的词. 所以它的性能只是介于两者之间.

3.3 基于字符树的分词扫瞄算法时间复杂度

基于字符树的分词扫瞄能够完全消除无效查询问题. 无论要处理的字符串是多少长, 它从字符串首字开始逐字与字符树对比, 一旦发现字符树的路径已经是“尽头”了, 则查询结束, 然后提取现有的最长的词. 它的时间复杂度取决于待分词字符的总长度, 与词库中最长词的长度无关. 所以, 它的分词算法的时间复杂度为 O(n), (n 为待分词总字符数). 采用字符树词库进行分词不仅消除了无效查询问题, 而且它对于利用回溯法[13]进行多种分词方案的选择时具有得天独厚的优势.

4 实验结果比较

实验环境: 操作系统是 Win10, CPU 为 Intel® Core™ i5-6500 CPU @3.20 GHz;内存大小为 4 GB. 为了区别各类词库的查询性能与内存占用情况, 本文编写了 6 个词库类, 如表 4 所示, 它们分别依次对同一个词库文件进行加载, 对同一个文本文件进行分词, 查看加载词库所用时间, 占用内存, 以及分词所花费时间等数据. 内存占用量用加载词库前程序占用总内存与加载词库后程序占用总内存的差作为取值, 内存占用信息通过操作系统提供的 GetProcessMemoryInfo 函数获取. 加载词库所需时间与对文件进行分词所需时间分别是在开始前用 clock() 函数获取时间, 结束后再次用该函数获取时间, 用它们的差值作为花费时间, 各 trie 索引树法的散列表的装载因子统一设为 0.5. 实验取得数据分析表见表 5 所示.

表 4 各词库类的数据结构及实现技术

序号	类对象	数据结构	实现技术
1	MySet	整词表	采用整词二分查找
2	Trie2	2 字 trie 树+余词表	前 2 字分级散列+余词折半法
3	Trie3	3 字 trie 树+余词表	前 3 字分级散列+余词折半法
4	Trie4	4 字 trie 树+余词表	前 4 字分级散列+余词折半法
5	DichotomyTree	单字符树	逐节点 (字) 折半法
6	ClusterTree	多字符树	逐节点 (字) 折半法

4.1 内存占用情况

本文提出的单字符树词库内存占用最小, 整词二分法词库最大; 而对 trie 索引树词库的内存占用情况是级数越多, 内存占用却越小. 原因是 trie 索引树分的级数越多, 就会有更多的共同的前缀字符被省略, 而“余词”数量也会大幅度减少, 所以占用内存会越少. 传统的整词词库占用内存最大, 最主要的原因是字符串对象的创建与排序会造成大量的“内存碎片”, 从而浪

费内存. 而多字符树与单字符树相比, 性能接近相同, 但内存占用却是多字符树更大, 原因是多字符树的每个节点多了一个指向字符数组的指针, 而且各节点中除第一个字符保存在节点内, 其余的字符都是组合成数组“挂接”在节点上, 需要另外的存储空间. 所以它的内存占用会比单字符树大得多.

表 5 实验数据统计表

序号	类对象	加载时间 (ms)	占用内存 (MB)	分词时间 (ms)
1	MySet	12 810	39	1140
2	Trie2	7811	38	107
3	Trie3	7192	32	50
4	Trie4	6960	17	46
5	DichotomyTree	11 400	7	34
6	ClusterTree	12 067	18	33

通过软件采集实验所用的词库文件获得词库总条数为 279 095, 总字符数为: 821 105. 平均每个词含字符数为 2.94 个. 而加载成单字符树后产生的节点数为 405 534, 相当于词库中的一半字符可以省略, 平均一个词仅占 1.45 个节点数. 单字符树一方面省略了共同前缀的字符; 另一方面, 也是更重要的是由于它的节点数据结构单一, 大小一致, 在构造字符树完成后可以将所有字符节点从“分散”状态“收集”到一张线性表中, 释放掉原来的所有节点. 这一过程会释放大量的“内存空闲碎片”, 从而减少内存占用. 它的内存占用只有整词词库 MySet 的占用内存的 1/5.5.

4.2 分词性能比较

分词速度最快的是本文提出的字符树分词法, 其次是 trie 索引树分词法, 最差是整词二分法. 字符树的分词速度大约是整词折半法的 35 倍. 而对于 trie 索引树词库的性能情况是级数越多, 分词速度越快.

原因分析: 字符树消除了分词过程中的无效查询的问题, 它同时在扫瞄算法与词库查询两个方面都具有最突出的优越性, 所以总体分词速度是最快的. 整词分词法由于在扫瞄算法上存在大量的无效查询, 而它的词库查询时间复杂度又是最高的, 所以它的分词速度最慢. 而 trie 索引树可以部分消除无效查询, 同时它的词库查询访问性能是介于字符树与整词法之间, 所以它的分词性能也处于两者之间. 另一方面, 不同级数的 trie 树性能也不同, trie 索引树级数越多, 也就是深度越深, 它的分词性能越好, 越接近字符树, 而级数越少, 则其性能越差, 越接近整词分词法.

总之 trie 索引树的分词性能是介于单字符树与整词法两者之间, 当它的级数增大则向字符树方向演变, 当它级数减少则向整词词库方法演变, 整词法与字符树法是 trie 树的两个极端情况.

5 结束语

本文从基于词库的中文分词思想出发, 对现有分词技术进行分析对比, 提出字符树的分词技术, 解决了中文分词中的无效查询问题. 首先它改进了词库查询方式与分词扫瞄方式, 大大减少了运算的复杂度, 减少运算量, 尤其在大规模词库中显示出具有的巨大优势; 其次, 单字符树词库占用内存小, 大大降低了程序的空间复杂度. 无论在时间复杂度还是空间复杂度, 单字符树都是一种占据绝对优势的, 在大数据处理中文信息时值得推广的中文分词技术.

参考文献

1 邓丽萍, 罗智勇. 基于半监督 CRF 的跨领域中文分词. 中文信息学报, 2017, 31(4): 9–19. [doi: 10.3969/j.issn.1003-0077.2017.04.003]

2 金宸, 李维华, 姬晨, 等. 基于双向 LSTM 神经网络模型的中文分词. 中文信息学报, 2018, 32(2): 29–37. [doi: 10.3969/j.issn.1003-0077.2018.02.004]

3 胡婕, 张俊驰. 双向循环网络中文分词模型. 小型微型计算机系统, 2017, 38(3): 522–526.

4 张义, 李治江. 基于高斯词长特征的中文分词方法. 中文信息学报, 2016, 30(5): 90–93.

5 丁洁. 基于层次分析法的中文分词算法改进. 信息技术, 2016, (10): 191–193.

6 姚兴山. 基于 Hash 算法的中文分词研究. 现代图书情报技术, 2008, (3): 78–81. [doi: 10.3969/j.issn.1003-3513.2008.03.014]

7 蔡蕊. 一种改进的基于 Hash 的中文分词算法研究. 福建电脑, 2010, 26(2): 69–70. [doi: 10.3969/j.issn.1673-2782.2010.02.048]

8 刘超, 王伟东. 基于双哈希词典机制中文分词的研究. 信息技术, 2016, 40(11): 152–156.

9 刘勇, 魏光泽. 基于双字哈希结构的最大匹配算法机制改进. 电子设计工程, 2017, 25(16): 11–15. [doi: 10.3969/j.issn.1674-6236.2017.16.003]

10 [10 熊志斌, 朱剑锋. 基于改进 Trie 树结构的正向最大匹配算法. 计算机应用与软件, 2014, (5): 276–278. [doi: 10.3969/j.issn.1000-386x.2014.05.070]

11 陈之彦, 李晓杰, 朱淑华, 等. 基于 Hash 结构词典的双向最大匹配分词法. 计算机科学, 2015, 42(11A): 49–54.

12 杨进才, 陈忠忠, 谢芳, 等. 基于汉语拼音首字母索引的混合分词算法. 计算机系统应用, 2016, 25(4): 221–225.

13 曹菲, 聂文惠, 陈伟鹤. 基于 Hash 的正向回溯算法的改进. 信息技术, 2017, (11): 167–171.