基于 FP-参考树/表的频繁模式挖掘算法*)

石 巍 傅 彦
(电子科技大学计算机科学与工程学院 成都 610054)

摘 要 通分析 FP-growth 算法中包含的冗余操作,引入数据结构 FP 参考树/表,改变 FP-growth 算法中条件模式基的存储和生成方式,提出了新的 FPRSG 算法,高效地解决了频繁模式挖掘问题。理论分析与实验结果表明,FPRSG 算法优于 FP-growth 算法。

关键词 关联规则,频繁模式,FP 参考树/表,FP 参考收缩/增长算法,条件模式基

Algorithm for Frequent Pattern Mining Based on FP-Reference-Tree/List

SHI Wei FU Yan
(Department of Computer Science and Engineering, UESTC, Chengdu 610054)

Abstract In order to resolve frequent pattern mining problem efficiently, redundant operation and temporary data in FP-growth algorithm are analyzed, data structure FPR-tree and FPR-list are imported, the method of conditional pattern base generation and storage in FP-growth algorithm are improved, and a new FPRSG algorithm is presented. Theoretical analysis and experiment result both show that FPRSG is more efficient than FP-growth.

Keywords Association rule, Frequent pattern, FP-Reference-Tree/List, FPRSG algorithm, Conditional pattern base

1 引言

给定一个事务数据库和最小支持度阈值,从中找出完整的频繁模式集的问题就是频繁模式挖掘问题。频繁模式挖掘是挖掘关联规则、序列模式、最大模式、显露模式等方法的基础。

为解决频繁模式挖掘问题,Agrawal 等人在文[1]中提出了 Apriori 算法,其他人在其基础上相继提出了 DHP、PMD、PARTITION、Sampling、PIUA、FMA 等算法。但这些算法的共同缺陷是在它们的运行过程中都需要产生大量候选集,导致算法的效率不高。

Han Jiawei 等人在文[2]中提出频繁模式增长(FP-growth)算法,它采取如下分治策略:将提供频繁项集的数据库压缩到一棵频繁模式树(FP-tree),但仍保留项集关联信息;然后将这种压缩后的数据库分成一组条件数据库,每个关联一个频繁项,并分别挖掘每个数据库。

与 Apriori 算法相比,FP-growth 算法有极大的性能提高。其原因主要在于 FP-growth 算法引入高度压缩的数据结构 FP-tree,避免多次扫描数据库,并引入基于 FP-tree 的频繁模式片段增长方法,不需要产生大量候选集。然而,FP-growth 算法仍然有改进的余地。本文通过分析 FP-growth 算法,找出其中对性能影响较大的冗余操作并加以改进,从而提出 FPRSG 算法。

2 频繁模式增长算法分析

FP-growth 算法是在一棵高度压缩的 FP-tree 上进行挖掘的,其主要过程为:在给定的事务数据库上进行两次扫描,构造一棵初始 FP-tree(直接由给定的事务数据库生成的 FP-tree),这棵 FP-tree 只存储按支持度排序的频繁项,通过合并含有完全相同的频繁项集的事务,且使含有部分相同频繁项的事务共享前缀,实现对数据库的压缩。此后的挖掘就在这棵 FP-tree 上进行:由初始后缀模式(长度为 1 的频繁模式)开始,通过连接后缀模式与 FP-tree 的频繁项逐步增长模式,根据 FP-tree 构造每个后缀模式的条件模式基(由与某后缀模式同时出现的前缀子路径的集合组成的子数据库)及条件 FP-tree,并递归地在该树上进行挖掘。

通过对 FP-growth 算法的分析,可以得到以下结论:

引理 若不考虑计数值和节点链,则根据一棵 FP-tree 构造的条件模式基可以用此 FP-tree 的子树表示。

证明:令后缀模式 K 的条件模式基 B 是根据某 FP-tree(记为 CT)直接构造。由条件模式基的定义可知,K 的条件模式基就是在 CT 中 K 的前缀子路径集合,即 B 中的每条事务 BP 都对应于 CT 上一条后缀为 K 的路径 P 的前缀子路径,且 BP∪K=P。这些路径的集合形成 CT 的一棵子树。故根据一棵 FP-tree 构造的条件模式基可以用此 FP-tree 的子树表示。

如果给这棵子树附加上计数值和节点链,那么除了项层次并未按支持度排序外,这棵子树也是一棵完整的条件 FP-tree,可以在其上递归增长频繁模式。这样得到的所有条件 FP-tree 都成为初始 FP-tree 的子树。

3 FPRSG 算法

3.1 FPR-tree 和 FPR-list

既然所有的条件 FP-tree 都可以成为初始 FP-tree 的子树,条件 FP-tree 的所有路径信息都可以从初始 FP-tree 中取得,那么 FP-growth 算法对每个条件模式基都生成条件 FP-

*)基金项目:本文受国家自然科学基金(14076006)资助。石 巍 硕士研究生,主要研究方向:计算智能,数据挖掘;傅 彦 教授,主要研究方向:计算智能,数据挖掘。

tree，独立地保存路径模式和计数值，这里含有冗余操作。如果将整个事务数据库存储到树形结构的路径模式和条件模式基的路径及计数值分开保存，用树上的单个节点表示从此节点到根节点的路径模式，用节点的集合表示路径集合从而表示一个子数据库，就可以简化路径信息的存储。为此，引入频繁模式参考树和频繁模式参考表。

频繁模式参考树（Frequent Pattern Reference Tree，FPR-tree）有一个标记为“null”的根节点，其子节点为项前缀子树的集合；每个项前缀子树的节点有 2 个域：项名 item 和父节点指针 parent。Item 记录该节点代表的项的名字，Parent 指向该节点的父节点。

频繁模式参考表（Frequent Pattern Reference List，FPR-list）是一个列表，每个表项由两个域 item 和 nodes 组成。项名 item 是该表项代表的项的名字，同项节点集 nodes 是一个数组，其元素有两个域：节点指针 node 指向 FPR-tree 中的一个作为事务后缀的项名为 item 的节点，计数值 count 记录以此节点为后缀的事务条数。

FPR-list 引用 FPR-tree 上的多个节点，用这些节点到根节点的路径表示事务，同时记录这些路径的支持度，从而表示一个事务集。若这个事务集是某后缀模式 K 的条件模式基，则此 FPR-list 称为 K 的条件 FPR-list，记作 FPR-list | K。FPR-list 在 FPR-tree 上标记出的带支持度的子树称为 FPR-list（在 FPR-tree 上）的标记子树。FPR-list | K 的标记子树相当于按整个数据库中事务频繁项的支持度排序的条件 FP-tree | K。记录数据库全部事务信息的 FPR-list 称为初始 FPR-list。

算法 1：从给定的事务数据库建立一棵 FPR-tree 以及初始 FPR-list。

输入：一个事务数据库 DB 和一个最小支持度 min_sup。

输出：与 DB 对应的 FPR-tree 和 FPR-list。

步骤 1：扫描 DB 一遍，得到每个项的支持度和按支持度降序排列的频繁项集合 I。

步骤 2：创建 FPR-tree 的根节点 R，将其标记为“null”。创建空的初始 FPR-list L。对 DB 中的每条事务 T 作如下处理：选出 T 中的频繁项并按 I 的顺序排序。把 T 中排序后的频繁项列表记为[p|P]，其中 p 是第一个元素，P 是列表的剩余部分。调用 insert_tree([p|P],R)。

函数 insert_tree([p|P],N)的运行如下：令 S 为 N 的子节点，使得 S.item=p.item。如果 S 不存在，则创建之。如果 P 非空，则递归调用 insert_tree(P,S)，否则调用 insert_list(L,S)。

函数 insert_list(L,N)的运行如下：令 E 为 L 中满足 E.item=N.item 的表项。若 E 不存在，则创建之。若 E 的 nodes 数组中已有对 N 的引用，则将其 count 域值增加 1；否则在 nodes 数组中增加一个新元素并令其 node 域指向 N，count 域值为 1。

表 1

TID	项 ID 列表	TID	项 ID 列表
T100	I1,I2,I5	T600	I2,I3
T200	I2,I4	T700	I1,I3
T300	I2,I3	T800	I1,I2,I3,I5
T400	I1,I2,I4	T900	I1,I2,I3
T500	I1,I3		

FPR-tree 与初始 FPR-list 的构造过程较为简单，此处不再详述。若给定表 1 的事务数据库作为输入，并假定最小支持度为 2，则算法 1 建立的 FPR-tree 和初始 FPR-list 如图 1 所示。

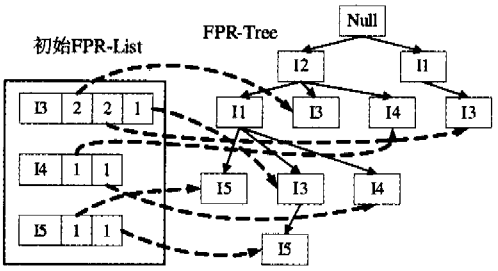

图 1

值得注意的是，FPR-list 中 nodes 域引用的节点是数据库或条件模式基中事务的后缀节点，而不仅仅是其标记子树的后缀节点。如图 1 中路径$\langle I_2, I_1, I_3, I_5\rangle$的 I_3 节点不是叶节点，但事务 T900 包含的频繁项$\langle I_1, I_2, I_3\rangle$按支持度排序后以 I_3 为后缀，所以 I_3 节点也被引用。

3.2 FPR-list 的收缩

挖掘过程中，需要按项层次从低到高遍历条件模式基。从效率考虑，希望能直接由 FPR-list 每项的节点数组取得标记子树中该层的所有节点。然而由于 FPR-list 是用后缀节点表示路径，除最后一项以外的其他项均未记录标记子树中包含该项而又不以其为后缀的事务的节点，是不完整的。这就需要在处理一层节点之前预先将这层的所有节点加到 FPR-list 对应项的 nodes 域中。这可以通过收缩（Shrink）操作完成。

FPR-list 的收缩就是在从后往前逐项处理当前 FPR-list 中每项 nodes 域引用的所有节点，以增长后缀模式并用节点的前缀子路径集构造新的条件模式基时，节点的父节点与计数值不但要加入到新后缀模式的条件 FPR-list，还要加入到当前 FPR-list 的对应项中，以保证其前缀子路径中各项引用节点的完整性。完整性的证明如下：

对于 FPR-list 标记子树中一条表示事务的路径 T，T 的后缀节点在创建 FPR-list 时就已被对应项引用，其支持度为 1；在处理 T 中的任一节点 M 所在项层次时，M 的父节点随着收缩加入到 FPR-list 更高层次的项引用数组中，故单事务路径 T 上的所有节点都将得到处理。又由于多条事务在 FPR-list 中具有可叠加性，因此收缩操作可以保证在处理 FPR-list 的任一项层次时，该项层次都已经包含标记子树中对应项层次的所有节点。

FPR-list 的表项处理完毕之后就不再使用，可以将其删除，故收缩操作的实质是从条件模式基的事务集中按支持度升序依次移除每个项，表现为标记子树往根节点逐层收缩。

3.3 在 FPR-tree 和 FPR-list 上挖掘频繁模式

通过以上分析，我们提出基于 FPR-tree 和 FPR-list 的 FPRSG（Frequent Pattern Reference Shrink-Growth）算法。

输入：由算法 1 建立的 FPR-tree 和初始 FPR-list。

输出：所有的频繁项集。

步骤：调用 FPR-Grow(FPR-tree，初始 FPR-list，null)。

下面是函数 FPR-Grow 和 List-Grow 执行过程的伪码描述：

Procedure FPR-Grow(Tree，List，α)

1. if List 中的所有表项都只有一个元素并且其对应节点都在 Tree 中的一条路径 P 上 then {

2. 对 List 从表尾到表头的每个表项中的唯一元素(记为 elem)，调用 List-Grow (List, elem. node. parent, elem. count)；

3. 对 P 中节点的每种组合(记为 β)，产生频繁项集 β∪α，其支持度为 β 中节点的最小支持度；}

4. else 对 List 从表尾到表头的每项(记为 entry){

5. 计算出 entry. nodes 域中所有元素的计数值之和，记为 s；

6. 若 s 小于最小支持度，则调用 List-Grow(List, elem. node. parent, elem. count)进行收缩；移除 entry；转到步骤 4；

7. 产生一个模式 β=α∪ entry. item，其支持度为 s；

8. 创建后缀模式 β 的空条件 FPR-list(记为 SubList)；

9. 对 entry 的 nodes 域中的每个元素(记为 elem)，调用 List-Grow(SubList, elem. node. parent, elem. count)构造 β 的条件 FPR-list，若 elem. node. parent 不为 null 则调用 List-Grow(List, elem. node. parent, elem. count)进行收缩；

10. 移除 entry；若 SubList 不为空，调用 FPR-Grow (Tree, SubList, β)；}

Procedure List-Grow(List, tree-node, trans-count){

1. 取得 List 中与 tree-node 项名相同的表项 entry，若不存在则新建该表项；

2. 取得 entry 中指向 tree-node 的元素 elem，若不存在则新建该元素并令其 node 域指向 tree-node，count 域为 0；

3. 将 elem 的 count 域增加 trans-count。}

下面仍以表 1 为例分析算法的执行过程。将算法 1 得到的 FPR-tree 和初始 FPR-list 作为算法的输入，按从表尾到表头的顺序考察初始 FPR-list 中的每个表项。对表项 I_5，产生频繁模式 I_5:2 并创建 FPR-list|I_5。I_5 的 nodes 数组中有两个元素，其计数值都为 1，节点在 FPR-tree 上的父节点分别为 I_1 和 I_3。这两个节点就是 I_5 的条件模式基中事务的后缀节点。为构造 I_5 的条件模式基，将这两个节点加入 FPR-list|I_5。为进行收缩，将这两个节点加入初始 FPR-list，这需要在初始 FPR-list 中增加一项 I_1，并在 I_1 的 nodes 域增加一个元素，指向相应的父节点并设置支持度为 1，I_3 域中已经有对另一父节点的引用，故只将其支持度增加 1。FPR-list|I_5 只有单路径，可以直接生成包含 I_5 的频繁模式 I_2 I_5:2、I_1 I_5:2 和 I_2 I_1 I_5:2。这时可以移除初始 FPR-list 中的 I_5 项，使标记子树收缩到 I_4 层。接下来处理初始 FPR-list 的 I_4 项。以此类推，对初始 FPR-list 中所有项进行处理。

图 2 是处理完 I_5、I_4 项后收缩到 I_3 的初始 FPR-list 和后缀模式 I_3 的条件 FPR-list。

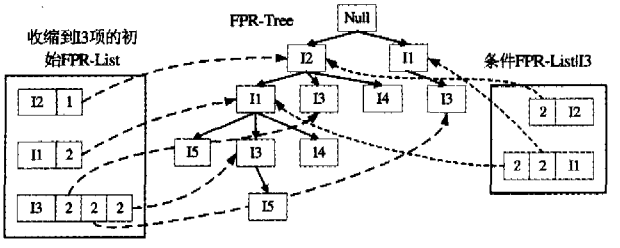

图 2

4 性能分析与测试

FPRSG 算法对 FP-growth 算法的改进，在于对条件模式基的处理在空间和时间效率上的提高。

在空间效率上，在处理事务数据库时，FP-growth 算法在挖掘过程中递归地为不断增长的后缀模式创建完整的条件 FP-tree，这导致在处理后缀模式$\langle I_1, I_2, I_3, \cdots, I_k\rangle$时，内存中需要保存条件 FP-tree|$\langle I_1, I_2, I_3, \cdots, I_k\rangle$，条件 FP-tree|$\langle I_2, \cdots, I_k\rangle$直到条件 FP-tree|$\langle I_k\rangle$等 k 棵条件 FP-tree，而 FPRSG 算法只需保存同样个数的条件 FPR-list。由于 FPR-tree 为全局共享，条件 FPR-list 仅记录条件模式基中事务的后缀节点，其内存占用一般比等价的条件 FP-tree 小得多，并且随着条件 FPR-list 的收缩，其占用的空间还将进一步减小。这提高了内存空间的使用效率，使相同大小的内存可以容纳更大的事务集。

在时间效率上，FP-growth 算法在构造条件 FP-tree 时需要两次扫描条件模式基并向树中插入条件模式基里每条不重复的事务，而 FPRSG 算法在构造条件 FPR-list 时只需向两个 FPR-list 中分别插入同样个数的节点。虽然 FPR-list 的项并未按支持度排序，但由于 FP-growth 算法中排序的主要优点在于减少条件 FP-tree 的节点数，而 FPRSG 算法中 FPR-tree 的节点由所有 FPR-list 共享，故排序与否对 FPRSG 算法的性能影响很小。总的来说 FPRSG 算法的时间效率也有较大的提高。

为实际测试算法的性能，采用文[1]的数据合成方法，合成一个事务数据库 T25. I20. D100K，项数为 10K。分别测试 FPRSG 算法与 FP-growth 算法在不同事务数与支持度阈值情况下的运行时间。实验平台的操作系统为 Windows XP，CPU 为 P4 1. 6G，内存为 512M。测试程序使用 Java 编写。运行时间随支持度阈值变化的测试结果见图 3，支持度阈值固定为 0. 5%时运行时间随事务数变化的测试结果见图 4。实验结果显示 FPRSG 算法在不同事务数和支持度阈值情况下的性能与扩展性都优于 FP-growth 算法。

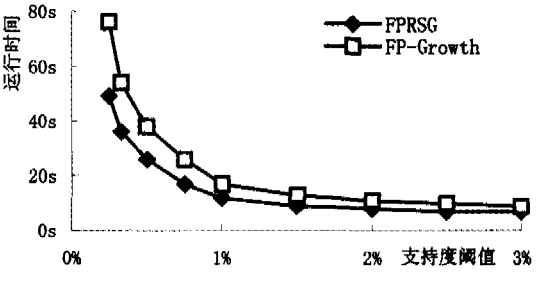

图 3 支持度阈值-运行时间

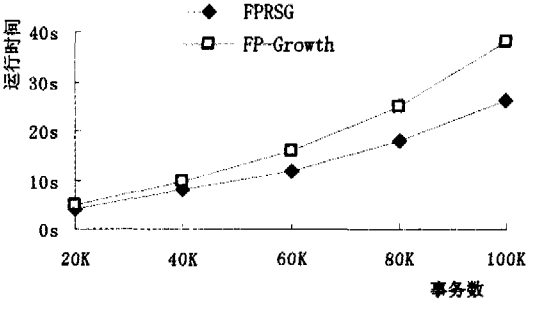

图 4 事务数-运行时间

结束语 本文针对频繁模式挖掘问题，引入 FPR-tree 和 FPR-list 数据结构，提出了 FP-growth 的改进算法—FPRSG，通过改变事务数据库与条件模式基的存储方式，减少了处理

过程占用的内存空间与处理时间，提高了频繁项集的挖掘效率。实验结果表明 FPRSG 算法优于 FP-growth 算法。

FPRSG 算法是从物理存储方面对 FP-growth 算法改进，完全可以将其与算法逻辑方面的改进(如对生成条件模式基必要性的判断和裁剪)结合起来，进一步提高频繁模式挖掘的效率。

参 考 文 献

1 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. 1994 Int. Conf. Very Large Data Bases(VLDB'94)[C], Santiago, 1994. 487～499

2 Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Proc. of 2000 ACM SIGMOD Int. Conf. on management of data [C], Dallas, USA, 2000. 1～12

3 Han J W, Pei J, Yin Y. Mining partial periodicity using frequent pattern trees [R]. Canada: Simon Fraser University: [Computing Science Technical Report: TR-99-10]. 1999

4 Mining generalized association rules [A]. In: Proc. of the 21st Int Conf. on Very Large DataBase [C]. Zurich, Switzerland, 1995. 407～419

5 王晓峰，王天然，赵越. 一种自顶向下挖掘长频繁项的有效方法. 计算机研究与发展，2004，41(1)：148～155

6 闫炜，崔杜武，付长龙. 基于幂集的关联规则挖掘算法研究. 计算机工程与应用，2004(1)：192～194

7 孟祥萍，王华金，王贤勇，任纪川，鞠传香. 基于改进 FP-树的最大模式挖掘算法. 计算机工程与应用，2005，14：179～181

8 陈安龙，唐常杰，陶宏才，元昌安，谢方军. 基于极大团和 FP-Tree 的挖掘关联规则的改进算法. 软件学报，2004(8)：1198～1207

(上接第 205 页)

表，发现有"school"可能匹配，然后查到 PREFIX 表时才知道不匹配。算法改进之后，在第二次匹配过程中，$m > 2$，每次至少检查两个字符，发现"*st*"与"*sc*"不匹配，我们不会转而去查 HASH 表和 PREFIX 表。这样就减少了很多没有意义的比较，大大降低算法匹配的时间。

4.2 算法性能分析

下面，我们对 Wu-Manber 算法及其改进算法进行测试比较。

首先，我们从英文语料中取 100 个长度小于 6 个字符的模式串，测试这 100 个模式串当中出现不同个数的单字节字符串时所用时间，如图 9 所示。

其次，我们从中文语料中取 100 个长度小于 6 个汉字的模式串，测试这 100 个模式串当中出现不同个数的单汉字字符串时所用时间，如图 10 所示。

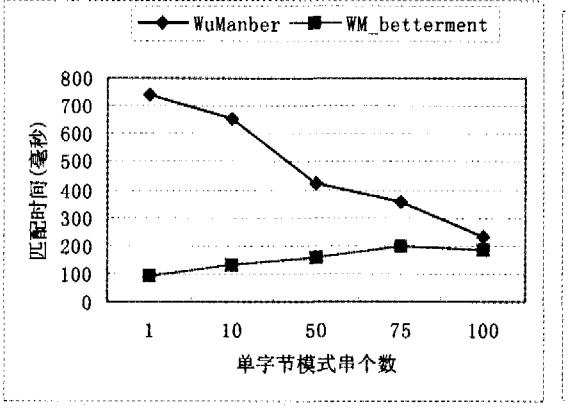

图 9 比较结果图(英文)

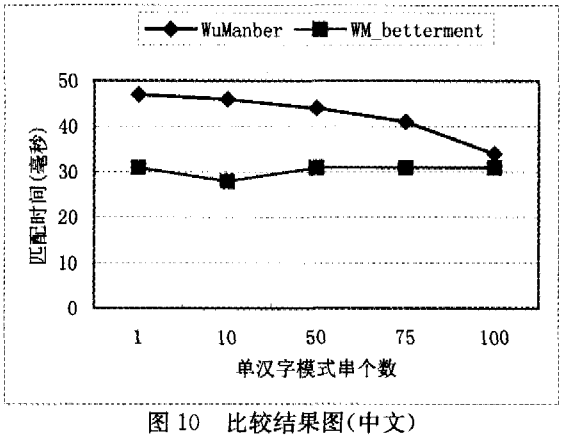

图 10 比较结果图(中文)

从图 9、图 10 可以看出，改进后的 Wu-Manber 算法性能有明显提高，匹配速度快于传统 Wu-Manber 算法。特别是对于英文匹配，在单字节模式串出现个数较少时，匹配速度较原来提高了 5～8 倍。考虑到现实中，单字节(单汉字)模式串出现个数较少，所以 Wu-Manber 改进算法还是具有一定的实用价值。

结束语 本文研究并测试了 Wu-Manber 算法，发现在大多数情况下，算法都具有良好的性能。针对匹配过程中出现单字节(单汉字)模式串时会大大降低 Wu-Manber 算法速度的情况，在不影响算法总体性能的基础上对算法进行了改进，使得算法性能有了进一步的提高。

参 考 文 献

1 Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search. Communications of ACM, 1975, 18(6): 333～340

2 Commentz-Walter B. A string matching algorithm fast on the average: [Technical Report]. The University of Heidelberg: IBM Heidelberg Scientific Center, Sep. 1979

3 Wu Sun, Manber U. A Fast Algorithm for Multi-pattern Searching: [Technical Report]. The University of Arizona: The Computer Science Department, May1994

4 Muth R, Manber U. Approximate multiple string search. In: Proc. 7th Combinatorial Pattern Matching (CPM'96). LNCS 1075. 1996. 75～86

5 Crochemore M, Czumaj A, Gasieniec L, Lecroq T, Plandowski W, Rytter W. Faster practical multi-pattern matching. Inf. Process. Leu, 1999, 71(3-4): 107～113

6 Wu Sun, Manber U. Agrep: A Fast Approximate Patternmatching Tool. Usenix Winter Technical Conference, San Francisco, 1992

7 Wu Sun, Manber U. GLIMPSE: A Tool to Search Through Entire FileSystem. Usenix Winter Technical Conference, San Francisco, 1994

8 Boyer R S, Moore J S. A fast string searching algorithm. Communications of ACM, 1977, 20(10): 762～772

9 谭建龙. 串匹配算法及其在网络内容分析中的应用：[学位论文]. 中国科学院计算技术研究所，2003