基于后缀树的带有通配符的模式匹配研究

侯宝剑[1] 谢 飞[1,2] 胡学钢[1] 刘应玲[1,3] 王海平[1]
(合肥工业大学计算机与信息学院 合肥 230009)[1] (合肥师范学院计算机科学与技术系 合肥 230601)[2]
(中国科学技术大学物理学院 合肥 230026)[3]

摘 要 由于在生物序列分析、文本索引、网络入侵检测等领域的应用需求,带有通配符的模式匹配问题一直是研究的热点。针对已有的研究工作中通配符和长度约束具有较强的局限性问题,研究带有灵活通配符的模式匹配问题,其中通配符可以在模式的任意两子串间出现且可以指定灵活的长度约束。采用非线性数据结构——后缀树,设计了求解模式所有解的完备算法PAST。预处理阶段采用在线增量式算法构建具有文本先验知识的后缀树,搜索阶段结合动态规划的思想,逐个匹配模式中字符,最终得到完备解。在基因序列上的实验表明,PAST比其他算法具有更好的时间性能。

关键词 模式匹配,通配符,后缀树
中图法分类号 TP309 **文献标识码** A

Pattern Matching with Wildcards Based on Suffix Tree

HOU Bao-jian[1] XIE Fei[1,2] HU Xue-gang[1] LIU Ying-ling[1,3] WANG Hai-ping[1]
(School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China)[1]
(Department of Computer Science and Technology, Hefei Normal University, Hefei 230601, China)[2]
(School of Physical Sciences, University of Science and Technology of China, Hefei 230026, China)[3]

Abstract Pattern matching with wildcards is a hot research problem that can be used in biological sequence analysis, text indexing, network intrusion detection, and so on. Aiming at the problem that the wildcards have strong limitations in the existing research work, pattern matching with flexible wildcards was studied. The wildcards can appear between any two substrings and can be specified with flexible length constraints. The nonlinear data structure—suffix tree was used to design a completeness algorithm PAST. In the prepare process, an online incremental algorithm was used to build the suffix tree which has priori knowledge of the text. In the search phase, the idea of dynamic programming was used to match the characters of the pattern. Experiments on DNA sequences show that our method has better performances in time than the related matching algorithm.

Keywords Pattern matching, Wildcards, Suffix tree

1 引言

2009年爆发的甲型H1N1对人类社会造成了重大损失,因此相关组织机构通过生物信息分析工具(例如BLAST[1]),分析已有相关病毒的基因序列,进而帮助研制疫苗。带有通配符[2]的模式匹配问题是相关分析工作中的关键问题,同时也是文本挖掘[3]、时间序列挖掘[4]、数据流挖掘[5]等领域的基础问题。

已有的带有通配符的模式匹配问题的定义具有局限性,例如:模式中任意两个相邻字符间通配符数或者模式中通配符总数[6]是固定的,以及模式中任意两个字符之间必须存在通配符[8,16]等。为了打破限制,实现可以根据实际问题灵活地指定通配符位置以及长度约束,本文研究带有灵活通配符的模式匹配问题(Pattern Matching with Flexible Wildcards, PMFW)。通配符的灵活性主要表现在出现位置和长度约束上,即通配符可以出现在任意子串之间,而长度约束的上限和下限可以独立地指定。文献[7]提出通配符也可以出现在两子串间,但由于只能在模式中出现一次,因此该问题仍然没有PMFW问题的灵活性高。

闵帆等人在文献[8]中提出带有独立通配符的模式匹配问题,并设计和分析了算法PAIG,该问题与PMFW问题极其相似。PAIG算法在搜索的过程中通过建立二维表来求解模式在文本中的完备解,而不关注解的位置。PAIG算法考虑了模式的全局约束,且通配符出现在任意两字符间,同时,

到稿日期:2012-02-16 返修日期:2012-07-13 本文受国家"863"计划课题(2012AA011005),国家博士后科学基金(2012M511403),安徽省自然科学基金(11040606M134)及中央高校基本科研基金(2010HGXJ0714)资助。
侯宝剑(1986-),男,硕士生,主要研究方向为模式匹配与数据挖掘,E-mail:baojian_hou@126.com;谢 飞(1980-),男,博士,讲师,主要研究方向为数据挖掘与自然语言处理;胡学钢(1962-),男,教授,博士生导师,主要研究方向为人工智能与数据挖掘,E-mail:jsjxhuxg@gmail.com(通信作者);刘应玲(1972-),女,博士生,主要研究方向为模式匹配与人工智能;王海平(1986-),男,博士生,主要研究方向为模式匹配与人工智能。

PAIG算法没有对文本进行预处理，搜索过程中产生的文本信息不能在多次匹配过程中共享和重用，因此处理多个模式的时间效率并不理想。为了能更好地实现信息的共享和重用，我们采用了一种应用在DNA序列分析[9]、XML结构索引[10]等领域中的非线性数据结构——后缀树[11]。在已有模式匹配问题，例如在经典模式匹配问题和带有固定通配符数的模式匹配问题，以及带有独立通配符的模式匹配问题的研究工作中，后缀树及其衍生结构DAWG[12]得到了大量的应用。

PMFW问题的解随着模式特征的变化呈指数级增长[8]。我们认为PMFW问题定义的灵活性增加了问题求解的复杂性，这是造成解的规模为指数级的重要原因，因此逐个计算输出每个匹配解的位置并不可行。例如：字母表 $\Sigma=\{A\}$，文本长度为5000，模式为$(A[a,b])^kA$，模式中第 i 个位置 A 确定后，则文本中存在 $b-a+1$ 个位置作为第 $i+1$ 个位置 A 的可选位置，其中 $1\leqslant i\leqslant k$，依次类推，最终的解显然为指数级。由于每个匹配解的开始位置和结束位置受到文本长度、模式长度和间隔长度的约束，因此可以在多项式的时间里计算出所有的匹配次数。本文只关注匹配解的个数，而不关注解的位置，并从理论上证明了PAST算法可以得到完备解。在实际的DNA序列上做了充分的实验，结果表明，与已有相关匹配算法相比，PAST算法具有更好的时间性能。

本文第2节介绍相关的工作；第3节给出问题定义；第4节详细描述算法的步骤，并进行时间和空间复杂度分析；第5节给出实验结果及算法的对比分析；最后总结全文工作。

2 相关工作

Fisher和Paterson于1974年第一次从理论复杂性上对带通配符的模式匹配进行泛化[2]。随后，一些针对不同形式的带有通配符的模式匹配问题算法被提出[6,14,16,17]。

Cole等人[6]对问题的定义进行了一些改变，即将原问题中模式串两个相邻字符间的固定的通配符数改进为固定模式串中总共的通配符数，该问题定义的不足之处在于通配符的长度受到限定而不是在一个范围内。Kucherov等人[14]对该问题做了进一步改进，即允许模式串中两个相邻字符间有任意个通配符。很明显，这仍然不能满足实际应用对模式灵活性的需求。

Chen等人[16]提出启发式SAIL算法，处理的问题考虑了以下方面：通配符出现在模式任意两字符之间、模式的局部通配符长度约束和全局长度约束以及One-off条件。SAIL分为前向搜索和后向搜索两阶段，首次提出用left-most匹配策略在多项式时间内求解问题，最终得到解的出现位置，但结果不是完备的。

Wang等人[14]提出SMST算法，其利用后缀Trie结构处理的问题与闵帆等人[3]提出的问题类似，但不考虑全局长度约束，最终得到解的个数，而不关注解的位置。但是构建后缀Trie结构消耗的时间和空间太大。

武优西等人[17]考虑模式的间隙约束和一次性条件(One-off条件)，基于网树的结构采用了贪婪搜索双亲策略和最右双亲策略，提出了一种启发式算法。该算法与SAIL一样不是完备的，但可以获得更好的解且解的质量较其它算法有显著的提高。

上述相关模式匹配算法问题定义中主要考虑通配符出现位置、长度约束、解的完备性以及出现位置，基于这几方面，以下给出PMFW问题的形式化定义。

3 问题定义

长度为 n 的文本记为 $T=t_0\cdots t_i\cdots t_{n-1}$，令 $T_i^j=t_i\cdots t_j$ 称为 T 的子串，其中 $0\leqslant i\leqslant j\leqslant n-1$。显然 $T_0^i=t_0\cdots t_i$，$T_i^{n-1}=t_i\cdots t_{n-1}$，且 T_0^i 和 T_i^{n-1} 分别称为 T 的前缀和后缀。Σ 为一字符表，包含文本 T 中不相同的字符。例如，DNA序列的字符表为 $\Sigma=\{A,C,G,T\}$。

通配符(Wildcard)可以代表字符集中任意字符，记为 Φ。间隔约束 $g=[N,M]$表示通配符个数的范围，N 和 M 分别为间隔的上限和下限。$G=M-N+1$ 为间隔长度。

模式 $P=p_1g_1p_2g_2\cdots g_{m-1}p_m$ 是由子串和通配符构成的序列，p_i 为字符串，其中的字符属于字符表，g_i 是 p_i 和 p_{i+1} 之间的通配符范围，即间隔约束。在本文中，p_i 为任意长度，所有的间隔约束的上限和下限可以灵活指定。模式 P 中含有字符表中字符的个数称为模式 P 的长度，记为$|P|$，用 L 表示。不考虑通配符的个数，模式也可以简记为 $P=p_1p_2\cdots p_m$。

定义1 如果存在一个位置序列索引 $l_1,\cdots,l_i,\cdots,l_m$(其中 $0\leqslant i\leqslant m$，$1\leqslant l_i-|p_i|+1\leqslant l_i\leqslant m$，$l_i$ 为子串在文本中匹配的结束位置)，满足条件：

(1) $p_i=t_{l_i-|p_i|+1}\cdots t_{l_i}$

(2) $N_i\leqslant l_{i+1}-l_i-|p_i|\leqslant M_i$

则 $l_1,\cdots,l_i,\cdots,l_m$ 称为模式 P 在文本 T 中满足间隔约束的一次出现。文本中相应的子串称为匹配子串。

定义2 给定模式 $P=p_1g_1p_2g_2\cdots g_{m-1}p_m$，令 $P_i^j=p_x^{|p_i|}g_i\cdots g_{j-1}p_0^y$(其中 $0\leqslant x\leqslant|p_i|$，$1\leqslant y\leqslant|p_j|$)，称其为模式的子串。子串中我们关心的是模式的后缀，即 P_i^m。

定义3 给定文本 $T=t_0\cdots t_i\cdots t_{n-1}$，由 T 的所有后缀构建的Tire tree结构称为 T 的后缀树，记为SuffixTree(T)，树中每个节点存储 T 的一个子串。

4 算法设计与分析

4.1 算法描述

给定文本 T 和模式 P，P 中带有灵活通配符，我们旨在设计一个算法来求解出模式 P 在文本 T 中的所有匹配子串的个数。

基于文献[8]中提出的算法，我们做了一些改进，设计了BuildSuffixTree算法。算法1给出了具体框架。首先自左向右扫描文本，逐步把字符添加到后缀树中(Lines3－6)。当文本所有字符全部添加成功后，将一个不在字符表中的字符(伪代码中使用'$')添加到后缀树(Lines7)，保证了所构建的SuffixTree(T)中根节点到叶子节点的路径和 T 的后缀一一对应。最后，把已构建好的SuffixTree(T)中有孩子节点的节点的字符串的最后一个字符删除，同时计算节点的叶子数(Lines 8－11)。

算法1 BuildSuffixTree(T)

```
输入：文本 T
输出：SuffixTree(T)
1. 初始化节点数组 nodearray;
2. for i=0 to n－1 do
3.   for j=leaf Count to i do
4.     if(add(T[j],T[i])=false)
5.       break;
```

```
6. add('$',T[n-1]);//添加字符表外的一个字符
7. for e=0 to nodearray.length-1 do
8.   if(nodearray[e].child! =NULL)
9.     nodearray[e].right=nodearray[e].right-1;
10.        nodearray[e].leaf=NumberOfLeaf(e);//算叶子
11. return treeroot;
```

算法 2 是模式匹配的算法描述。首先初始化子问题列表和字符信息表(Lines 2-3)。自左到右逐步匹配模式中的字符。若 $P[i]$为字符表中的字符,则在本节点或者孩子节点中搜索匹配(Lines 6-9)。若 $P[i]$为间隔,则需要将本节点以及孩子节点中所有与 $P[i+1]$相等的字符信息提取出,并存入字符信息表 ChInfor。然后在子问题列表 SubProList 搜索与 ChInfor 中的每一条信息对应的子问题结果,若搜不到,则递归调用算法 2,搜索匹配 ChInfor 中的每一条信息对应的子问题,并将结果存入子问题列表 SubProList,此处采用了动态规划的思想来解决子问题重复计算的问题。

算法 2 MatPatt(treeroot,P)

```
输入:输入后缀树根节点 treeroot,模式 P
输出:模式 P 的所有匹配次数
1. varnode=treeroot;
2. 初始化子问题列表 SubProList;
3. 初始化字符信息表 ChInfo;
4. for i=0 to |P|-1 do
5. if(P[i]∈Σ)
6.   if(varnode 中字符串有未匹配的字符)
7.     search P[i] in varnode;
8.   else
9.     search P[i] in varnode.child;
10. else //P[i]为通配符
11.   if(varnode 中字符串满足长度约束)
12.     ChInfor=getchar();//提取等于 P[i+1]的字符
13.     for j=0 to ChInfor.length do//搜 P[i+1]并存储
14.       if(SubProList[ChInfor[j]]=NULL)
15.         SubProList=MatPatt(ChInfor[j],P^m_{i+1});
16.     clear(ChInfor);
17.   else
18.     ChInfor=getchar();//提取与 P[i+1]相等的字符
19.     for j=0 to ChInfor.length do//搜 P[i+1]并存储
20.       if(SubProList[ChInfor[j]]=NULL)
21.         SubProList=MatPatt(ChInfor[j],P^m_{i+1});
22.       clear(ChInfor);
23. return SubProList.matchresult;
```

4.2 算法完备性分析

PAST 算法可以得到 PMFW 问题的完备解。为了证明完备性,需要用到以下 3 条性质。

性质 1 给定文本 T、T 的子串 T_i^j 和后缀 T_i^{n-1},则 T_i^j 一定是 T_i^{n-1} 的前缀。

根据字符串的前缀和后缀的定义,可知性质 1 成立。例如:T=abcdefg,T_2^3=cd,T_2^5=cdefg,显然 cd 是 cdefg 的前缀。

性质 2 文本的后缀与后缀树中根节点到叶子节点的路径一一对应。

后缀树的构建算法[8]可以保证该性质的正确性。由性质 1 和性质 2 又可以得到性质 3。

性质 3 若 PMFW 问题存在解,则匹配子串在后缀树中必对应一段根节点开始路径。

算法从根节点开始逐步匹配模式中的字符,直到模式 P 的最后一个字符匹配成功。当遇到通配符时,则在满足长度约束范围内匹配通配符的下一个字符。性质 1 和 2 可以保证最后一个匹配字符所在的节点的叶子数即为匹配解。由于是逐个查找匹配,因此所有满足条件的路径均被搜索,并且最终得到完备解。

证明:假设算法得不到完备解,解集之外存在一个解。该解必对应文本中的一个子串 T_i^j,且满足模式 P 中的长度约束与 P 完全匹配。由性质 3 可知,T_i^j 必与后缀树中根节点开始的一段路径相对应。而所有与模式 P 的匹配子串对应的以根节点开始的路径均被搜索,且匹配成功,因此 T_i^j 对应的该路径必定不是以根节点开始的。又由性质 2 知,T_i^j 必不是文本 T 的后缀的前缀,这与性质 1 矛盾,因此假设不成立,PAST 算法得到 PMFW 问题的完备解,证明完毕。

4.3 算法复杂度分析

4.3.1 时间复杂度

算法预处理阶段采用改进后的后缀树构建算法,在线性时间里构建文本的后缀树,时间复杂度为 $O(n)$[11],n 为文本长度。算法搜索阶段逐步匹配模式中字符,如果该字符为间隔,并且在满足局部约束的范围内有满足条件的字符,则将未进行的搜索过程作为子问题来处理,即在子树中搜索子模式。若在局部范围内有多个满足条件的字符存在,它们对应重复子问题在分别计算时会重复计算更小的子问题。因此算法采用子问题表来解决该问题,即把已计算的子问题结果存入子问题表中,之后直接读取即可。算法在最坏的情况下,时间复杂度为 $O(n+L+G^m)$,其中 L 为模式的长度,G 为模式中的最大间隔,m 为模式中所有的间隔个数。

4.3.2 空间复杂度

算法预处理阶段构建的后缀树的节点与文本长度以及字符表有关,其数目与文本的长度为同一数量级[11],且存储在节点数组中。为了解决重复子问题,建立的 SubProList 表中的一项与一个节点相对应。因此节点数组的大小、SubProList 表的大小均与文本长度在同一数量级,所以算法的空间复杂度为 $O(n)$,n 为文本的长度。

5 实验及结果分析

实验环境:采用 JAVA 编程语言,编程环境:Eclipse,处理器为 Intel(R) Core(TM)2 Duo CPU E7500 @2.93GHz 2.94GHz,内存大小为 2.00GB,系统为 32 位 WIN7 操作系统。

以下 4 组实验考虑模式个数、子串个数、子串长度以及间隔长度 4 个参数,在同一数据集中与 PAIG 算法[8]做了对比。其中所使用的基因数据集下载于 NCBI[18],编号为 AB038490,长度为 131892。

实验 1 令子串个数为 4,子串长度为 4,间隔长度为 6,模式个数为 1、10、20、50、100、150、200、400、600、800、1000,实验结果如图 1 所示。

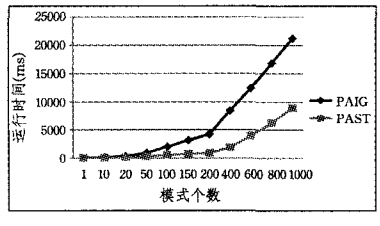

图 1 不同模式个数情况下 PAST 与 PAIG 算法的时间性能比较

PAST 算法中后缀树为一次构建多次使用，因此在同一文本中处理的模式越多，时间性能越优。从图 1 可以看出，随着处理模式个数的增加，PAST 算法使用的时间越来越少于 PAIG 算法。

实验 2　令模式个数为 100，子串长度为 4，间隔长度为 6，子串个数为 2、4、6、8、10、12、14、16、18、20，实验结果如图 2 所示。

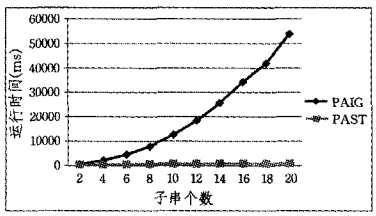

图 2　不同子串个数情况下 PAST 与 PAIG 算法的时间性能比较

由于 PAST 算法在后缀树中从根节点开始逐步匹配模式中的字符，因此当模式中某个字符匹配失败时，整个匹配过程会提前结束，而不需匹配模式中后一部分字符。随着子串个数的增加，模式在文本中的匹配解会越来越少，PAST 算法会提前结束大多数的匹配过程，因此子串个数的增加对 PAST 影响不大。而 PAIG 算法在不存在解的情况下仍需搜索，不能提前结束，因此子串个数的增加同时也会使得每一次的搜索时间增加。从图 2 可以看出，随着子串个数的增加，PAST 算法时间曲线几乎没有波动，而 PAIG 算法的时间曲线增长趋势明显。

实验 3　令模式个数为 100，子串个数为 4，间隔长度为 6，子串长度为 1、2、3、4、5、6、7、8、9、10，实验结果如图 3 所示。

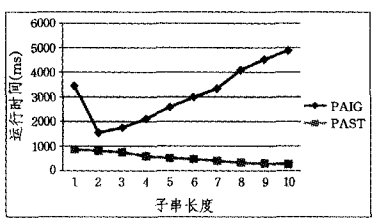

图 3　不同子串长度下 PAST 与 PAIG 算法的时间性能比较

PAST 算法的时间复杂度为 $O(n+L+G^m)$，若模式中无间隔，则时间复杂度为 $O(n+L)$。模式子串长度的增加，会造成间隔在整个模式中的比重减小，同时由于后缀树为一次构建，因此算法时间复杂度会逐步向 $O(L)$ 逼近。如图 3 所示，随着模式子串长度的增加，PAST 算法所用时间在减小，而 PAIG 算法所用时间逐步增加，优越性显而易见。

实验 4　令模式个数为 100，子串个数为 4，子串长度为 4，间隔长度为 0、1、2、4、6、8、10、12、14、16、18、20、22、24、26、28、30，实验结果如图 4 所示。

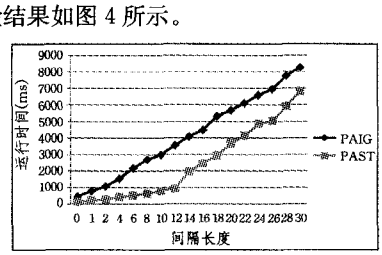

图 4　不同间隔长度情况下 PAST 与 PAIG 算法的时间性能比较

在子串长度固定而间隔长度很小的情况下，我们采用的子问题列表会有效地避免重复计算子问题，时间复杂度会远小于 $O(n+L+G^m)$。间隔长度的增大，会令子问题发生重复的机率大大减少，因此时间复杂度会向 $O(n+L+G^m)$ 逐步趋近。从图 4 中的时间曲线中可以看出，在间隔长度为 14 时，曲线出现了折点，说明此时运行时间比间隔长度为 12 时明显增加，但时间性能一直比 PAIG 算法优越。

结束语　本文基于后缀树，针对带有灵活通配符的模式匹配问题，利用动态规划思想设计了 PAST 算法，讨论了模式个数、子串个数、子串长度以及间隔长度 4 个参数对算法时间性能的影响。模式中通配符长度约束可以灵活指定，且可以出现在任意子串间，使得问题的定义在实际应用中更加合理。实验结果表明，PAST 算法在得到完备解的情况下，具有更好的时间性能。本文下一步工作将考虑约束条件下匹配解的位置的输出、带有 One-off 条件[16]的 PMFW 问题等。

参 考 文 献

[1] Altschul S F, Gish W, Miller W, et al. Basic local alignment search tool [J]. Journal of Molecular Biology, 1990, 215(3): 403-410

[2] Fischer M J, Paterson M S. String matching and other products [J]. Complexity of computation, Massachusetts Institute of Technology, 1974, 7: 113-125

[3] Don A, Zheleva E, Gregory M, et al. Discovering interesting usage pat-terns in text collections: integrating text mining with visualization[A]// Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM'07)[C]. New York: ACM, 2007: 213-222

[4] Yang Jiong, Wang Wei, Yu P S. Mining asynchronous periodic patterns in time series data [J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(3): 613-628

[5] Tanbeer S K, Ahmed C F, Jeong B-S, et al. Efficient frequent pattern mining over data streams[A]// Proceeding of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)[C]. California: ACM, 2008: 1447-1448

[6] Cole R, Gottlieb LA, Lewenstein M. Dictionary matching and indexing with errors and don't cares[A]// Proceedings of the 36th ACM Symposium on the Theory of Computing[C]. New York: ACM, 2004: 91-100

[7] Manber U, Baeza-Yates R. An algorithm for string matching with a sequence of don't cares [J]. Information Processing Letters, 1991, 37(3): 133-136

[8] Min Fan, Wu Xin-dong, Lu Zhen-yu. Pattern Matching with Independent Wildcard Gaps[A]// 2009 Eighth IEEE International Conference on Dependable[C]. 2009: 194-199

[9] 霍红卫，王小武. DNA 序列中基于适应性后缀树的重复体识别算法[J]. 计算机学报，2010，33(4)：747-754

[10] 张峰. 一种基于后缀树的时态 XML 索引研究[D]. 济南：山东大学，2010

[11] Ukkonen E. On-line Construction of Suffix Trees [J]. Algorithmica, 1995, 1(14): 249-260

[12] Blumer A, Blumer J, Haussler D, et al. Complete Inverted Files for Efficient Text Retrieval and Analysis [J]. ACM, 1987, 34(3): 578-595

[13] Gusfield D. Algorithms on Strings, Trees and Sequences-Computer Science and Computational Biology[M]. Cambridge: Cambridge University Press, 1997: 87-207

(下转第 194 页)

动态手势，表 2 为由各种基本手势组合成的各种类型的手语的具体含义。表 1 存储了 5 种静态手形，6 种基本轨迹，它们共组合成 30 种基本动态手势。表 2 为具体的各种手语及其含义的存储，首先有静态手语，如 a、b、c 分别代表“好”、“一”、“二”，简单手语 d1，即为手掌向前，表示的含义为“前进”，进而由几种简单手势组合成各种复杂手语，如 d5&d6，即为手掌向左、手掌向右，表示的含义为“不”，手语 d5&d6&d2 即为手掌向左、手掌向右、手掌向后，表示的含义为“不要”。以此类推，通过这样的不断组合，可以构成很多种复杂的手语，并存储其含义。

表 1　基本动态手势

轨迹 \ 手形	好(a)	一(b)	二(c)	五(d)	八(e)
向前(1)	好向前	一向前	二向前	五向前	八向前
向后(2)	好向后	一向后	二向后	五向后	八向后
向上(3)	好向上	一向上	二向上	五向上	八向上
向下(4)	好向下	一向下	二向下	五向下	八向下
向左(5)	好向左	一向左	二向左	五向左	八向左
向右(6)	好向右	一向右	二向右	五向右	八向右

表 2　手语含义表

手语	含义
a	好
b	一
c	二
…	…
d1	前进
…	…
d5&d6	不
…	…
d5&d6&d2	不要
…	…

建立基于索引结构的手势模板库，一方面可以节省存储空间，如在中国手语中，有很多不同含义的手语都会用到“手掌”这个基本手形，此时只需对手掌进行一次存储，然后对复杂手语进行手形标记的链接即可，如本数据库中手掌的标记为 d；另一方面，使用基于索引结构的手势模板库，可以进行高效的手势匹配与识别。如表 1 所列，在该手势模板库中共定义了 5 种静态手形，6 种动态轨迹（向上、向下、向左、向右、向前、向后），本手势模板库中共存储了表 2 中所列 6 种具体手语及其含义。对于静态手语可以直接搜索静态手势词库得出手势匹配结果，如静态手势“好”，直接在手语含义库中搜索 a，即可得到识别结果。对于手语“前进”的搜索，首先可以对手形进行搜索，得出静态手形为“手掌”，然后在对手势轨迹进行搜索匹配时，可以根据存储的具体手语缩小搜索范围，即在手语含义库中可知手掌向前、手掌向左及手掌向右是有具体意义的，在轨迹搜索时只需在“向前”、“向左”、“向右”中进行搜索，并得出手势轨迹结果，进而得出该手势为“手掌向前”，最后在手语含义库中进行搜索得出该手语含义为“前进”。

结束语　从手势搜索匹配过程中可以看出，基于索引结构的手势模板库的设计可以在很大程度上缩小搜索范围，提高搜索效率，实现高效的数据搜索，保证识别系统的准确性与实时性。在对该手势模板库进行设计时，分别按静态手形、运动轨迹及手语含义对基于手势词进行分类存储，可以简单地将其应用于静态手势的识别中，而且可以针对手势模板库的使用背景对各分类表进行相应的修改，如手势“八”在普通的手语环境中可定义为数字：“八”，而在作战手语中可将其含义修改为“枪”，在这种情况下只需修改手势语义表即可，修改简单，扩展性强。另外对于比较复杂的动态手语（连续动态手势），可以认为是由多个基本动态手势组成的，同样也可以使用该手势模板库。

本文也存在着一定的不足，如在对手语进行存储的时候，中国手语在表达含义上不仅仅使用手，有些手语的表达还结合了其他人体部位的表达，如“悲伤”的手语表达是一手虚握于胸部，并转动几下，脸露愁容，即结合了人脸的表情，有些手语的表达要结合手臂或双腿及整个身体的动作进行辅助，这样在对手语模板进行存储时，就需要考虑多方面的因素，在对手语进行识别时要综合考虑并得出其最后结果。另外，该手语模板库只是实现了单手的手语存储，对于双手的手语表达及存储没有充分的研究，其研究重点在于双手的定位与配合，可以通过对双手作标记的方法来实现，相信在未来的研究中可以取得进一步的成果。

参 考 文 献

[1] Alejandro J, Sebe N. Multimodal human-computer interaction [J]. Computer Vision and Image Understanding, 2007, 108(1/2): 116-134

[2] Pavlovic V I, Sharma R, Huang T S. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997, 19(7)

[3] 黄国范. 基于切线距离的手势识别[D]. 重庆：西南大学，2009

[4] 曹陶科. 基于光流计算和 DTW 算法的动态手势识别研究与实现[D]. 天津：天津师范大学，2009

[5] 于洋. 基于手形特征的静态手势识别[D]. 天津：河北工业大学，2007

[6] 郭兴伟. 基于视觉的手势识别算法研究[D]. 上海：上海海联学院，2003

[7] 中国聋人协会. 中国手语[M]. 北京：华夏出版社，2003

（上接第 180 页）

[14] Wang Chi, Wang Hao, Xie Zhao, et al. Exact String Matching With Variable Length of Don't Cares Based on Suffix Tree[A]// International Conference on Information Science and Technology, 2011[C]. Nanjing: IEEE Computer Society, 2011: 109-113

[15] Kucherov G, Rusinowitch M. Matching a Set of Strings with variable length don't cares [J]. Theoretical Computer Science, 1997, 178(1/2): 129-154

[16] Chen Gong, Wu Xin-dong, Zhu Xing-quan. Efficient string matching with wildcards and length constraints[J]. Knowledge and Information Systems, 2006, 10(4): 399-419

[17] 武优西，吴信东，江贺，等. 一种求解 MPMGOOC 问题的启发式算法[J]. 计算机学报，2011，34(8)：1452-1462

[18] NCBI National center for biotechnology information website [OL]. http://www.ncbi.nlm.nih.gov/