一种支持多种子近似串匹配的q-gram索引

孙德才[1] 王晓霞[2]
(渤海大学信息科学与技术学院 锦州 121013)[1] (渤海大学大学计算机教研部 锦州 121013)[2]

摘 要 如何在大型文本库中快速找出给定串的近似串是大数据时代要解决的关键问题。基于多种子的近似串匹配算法因匹配速度快而得到众多学者的青睐,但巨大的索引空间消耗也使其难以处理大型文本库。提出了一种支持多种子的q-gram索引结构,通过该索引能够快速地计算出给定任意长度连续种子的地址集合,解决了多种子近似串匹配算法中种子的数目和长度受存储空间限制的问题。实验数据显示,新索引方案成倍地减少了存储空间的消耗。实验结果表明,提出的索引方案在大数据环境下的多种子近似匹配中具有一定的优势。

关键词 大数据,近似串匹配,种子,q-gram索引,多种子索引
中图法分类号 TP391.3 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.053

Q-gram Index for Approximate String Matching with Multi-seeds

SUN De-cai[1] WANG Xiao-xia[2]
(College of Information Science and Technology, Bohai University, Jinzhou 121013, China)[1]
(Teaching and Research Institute of College Computer, Bohai University, Jinzhou 121013, China)[2]

Abstract How to find out all approximate strings of a given string from a big text database quickly is a key issue in the age of big data. Approximate string matching algorithm based on multi-seeds is researched for its fast searching speed. But it is difficult to process large text database due to its huge memory consumption. Here, a new q-gram index was proposed to solve the problem of multi-seeds used in approximate string matching algorithms. In the proposed index, the addresses of consecutive seed with arbitrary length can be computed out quickly. The experimental results demonstrate that the space consumption is decreased largely. As a result, the proposed index is of great practicality to deal with large database in the age of big data.

Keywords Big data, Approximate string matching, Seed, Q-gram index, Multi-seeds index

1 引言

随着互联网技术的飞速发展,大数据时代已来临,随着信息的急剧膨胀,巨大的信息库使得个人的接受能力严重超载。因此如何从这巨大的信息库中快速找出所需的近似信息一直是计算机科学研究的基本问题,即数据近似匹配问题[1]。近似串匹配能从大文本库中快速找出与给定查询近似的字符串,是解决文本数据近似匹配的关键技术。近似串匹配技术应用领域较为广泛,如生物信息学、文本检索、信号处理和模式识别等。

近似串匹配(Approximate String Matching, ASM)[2,3]是在一个相对较长的文本串中查找所有与给定的相对较短的模式串之间错误数不大于一定阈值的近似串,而错误数最常用的表示方法是编辑距离[4]。当文本库非常大,且要频繁采用不同模式串进行匹配时,基于索引的On-line近似匹配算法具有非常快的匹配速度[5]。近似串匹配中常用的索引结构有后缀树[6]、后缀数组[7]和q-gram索引等。q-gram索引[8,9]因其占用内存少、容易管理,且在等存储空间下索引的速度要快于后缀数组[10],因此在近似串匹配中被广泛采用。

近似串匹配算法中多采用定位查询串子串的方法,常称这些子串为种子。生物信息学中经典的基因序列比对算法BLAST[11,12]是一个启发式算法,它首先选取查询串中一些子串作为种子,然后通过定位这些种子在库中的位置,对被定位种子的区域进行扩展进行匹配串的查找。文献[13,14]把模式串分成$k+s$个种子,然后验证在文本串中出现至少s个种子且位置也正确的文本区域,如文献[13]中$s=1$,而文献[14]中$s>1$;文献[15-17]对文本串和模式串进行连续重叠q-gram拆分,称为连续种子,然后验证出现一定数目以上种子的文本区域;文献[18]从文本串中选取一部分种子,出现一定数目种子的文本范围将会被扩展和验证;还有文献[19,20]等基于留空种子的算法及相关优化,以及文献[21,22]中采用近似种子进行匹配的相关算法。

种子是由任意个字符构成且允许留空的字符串,因此受长度和字符变化影响其形式多种多样,即使固定长度且连续

到稿日期:2013-11-09 返修日期:2014-01-17 本文受2014年辽宁省博士科研启动基金计划(20141138),辽宁省社科联2014年度辽宁经济社会发展立项重点课题(2014lslktzdian-04),国家自然科学基金项目(61173142,61232016,61202462,61173141,61173136),辽宁省教育厅一般项目(L2013422)资助。

孙德才(1979—),男,博士,讲师,主要研究方向为近似串匹配、文本复制取证等,E-mail:sdecai@163.com;王晓霞(1977—),女,硕士,讲师,主要研究方向为近似串匹配、入侵检测等,E-mail:wxxsdc@163.com(通信作者)。

的种子也因其长度和各个位置字母的不同而数量巨大。近似串匹配中常采用单形式的种子进行匹配，如文献[15-17]中的固定长度为 q 的 q-gram 种子。近年来，为提高近似串的匹配速度和灵活性，基于多种子的匹配算法已成为研究的热点[19,20,23]，如 PatternHunter[19] 是一个基于双种子的近似串匹配算法，其匹配速度快且精度高，但其索引结构仅仅采用两个种子的独立索引的简单合并，因此该种索引空间消耗巨大，当采用多种子时不仅难以推广，更难应用到大型文本库的处理中。另外也有学者研究变长种子的索引结构[24,25]，但使用该类索引进行近似匹配时，种子的使用又受到索引结构的限制，不够灵活。

本文解决的主要问题是研究用于大型文本库近似串匹配的且支持多种子的索引结构。本文在 q-gram 索引结构的基础上提出了支持多种子的索引策略，通过该索引能够快速地计算出给定的任意长度的连续种子在文本库中出现的地址集合，减少了索引的内存空间消耗。本文第 2 节介绍了 q-gram 索引结构。第 3 节给出了任意种子的地址集合计算理论和算法；第 4 节析了索引的时间和空间消耗；第 5 节进行了相关实验并分析了实验结果；最后则给出了本文的结论。

2 支持多种子的 q-gram 索引结构

2.1 支持多种子 q-gram 索引结构

q-gram 索引[8,9]是一个倒排索引，其索引项为固定长度为 q 的字符串，即 q-gram。为方便基于多种子匹配算法的近似匹配，本文采用的 q-gram 索引包括文件顺序表、词汇表、倒排列表和遗漏种子 4 部分。遗漏种子是为防止计算中丢失种子的地址分配的一个字符数组 C，用于存储文本库中首 $q-1$ 个字符或尾 $q-1$ 个字符(见 3.1.2 节)。

文本库是众多文本文件的集合，如图 1(a)所示。这里把文本库中所有文件的内容在逻辑上按顺序依次连接，形成一个长文本串 T，如图 1(b)所示。

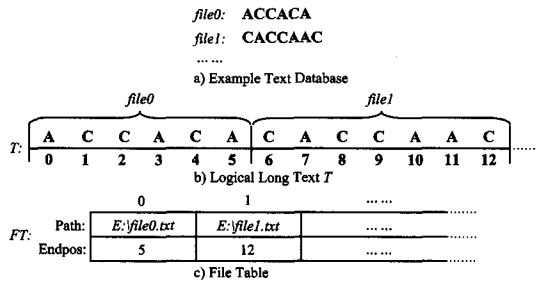

图 1 逻辑文本串 T 和文件顺序表

2.1.1 文件顺表

文件顺序表是一个长度为文本库中文件的总数的一维数组 FT，下标从 0 开始。文件顺序表中存储了文本库中每个文件的路径及其尾字符在逻辑文本串 T 中的结束位置，如图 1(c)所示。对于给定的任意文本范围 $T[a,b]$，定位其包含内容所属的文件及范围过程如下：

1)在 FT 中二分查找定位元素 i，使 $FT[i]$ 满足：$a \leqslant FT[i], i=0$ 或 $FT[i-1]<a \leqslant FT[i], i>0$。

2)从 i 位置开始向后顺序查找定位元素 j，使其满足：$b \leqslant FT[j], j=0$ 或 $FT[j-1]<b \leqslant FT[j], j>0$。

3)此时，如果 $i=j$，则该内容对应文件 i 中范围 $[a,b]$，$i=0$ 或 $[a-FT[i-1]-1, b-FT[i-1]-1], i>0$。如果 $i<j$，则该内容对应文件 i 的后部分，文件 j 的前部分，文件 $i+1$ 到 $j-1$ 的全部内容，文件 i 的后部分内容为 $[a, FT[i]], i=0$ 或 $[a-FT[i-1]-1, FT[i]-FT[i-1]-1], i>0$，文件 j 的前部分内容为 $[0, b-FT[j-1]-1]$。

2.1.2 词汇表

词汇表是文本库中所有不同的 q-gram 的集合，常采用 B$^+$ 树或哈希表存储结构。为能在词汇表中快速定位任意 q-gram索引项的位置，本文词汇表的结构采用哈希表存储。为确定哈希函数，首先为文本库字母表的每个字符建立和整数一一对应的映射，这里设字母表大小为 σ。若基因数据库中只有 4 个字符($\sigma=4$)，则可采用式(1)进行映射。

$$I(x)=\begin{cases}0, & x='A'\\ 1, & x='C'\\ 2, & x='G'\\ 3, & x='T'\end{cases} \tag{1}$$

因词汇表中索引项为 q-gram，本文采用 Karp-Rabin 函数[26]作为哈希函数，该函数能把任何一个长度为 n 的字符串映射成为唯一的整数，如式(2)。

$$H(S)=\sum_{i=0}^{n-1} I(S[i])\sigma^i \tag{2}$$

式中，S 为一个长度为 n 的字符串，因这里索引项为 q-gram，所以这里 $n=q$，$S[i]$是串 S 中的第 i 个字符。$H(S)$是串 S 的哈希值，其范围为 $[0,\sigma^q]$。对于连续重叠 $q-1$ 个字符的相邻 2 个 q-gram 项，第 2 个 q-gram 项的哈希值不需重复采用式(2)计算，而式(3)能在 O(1)的时间内获得其哈希值，如 $T[j+1, \cdots, j+q]$的哈希值能通过前一个 q-gram 计算得出。

$$H(T[j+1,\cdots,j+q])=I(T[j+q])+(H(T[j,\cdots,j+q-1])-I(T[j])\times\sigma^{q-1})\times\sigma \tag{3}$$

本索引中作为词汇表的哈希表长为索引项的总数 σ^q，因此这里分配一个长度为 σ^q 的一维数组作为哈希表，如图 2 所示。

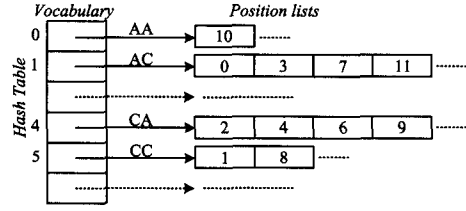

图 2 q-gram 索引实例的词汇表和倒排列表

2.1.3 倒排列表

词汇表中每个 q-gam 项后都连有一个倒排列表，用来存储该 q-gram 项在逻辑长文本串 T 中出现的地址集合，这里分配变长的一维数组作为倒排列表，如图 2 所示。

2.2 q-gram 索引创建

给定 q-gram 索引项的长度 q 值，创建索引只需一次完整地扫描文本库，预处理过程如下：

1)分配一个长度为 σ^q 的一维数组 HT 作为哈希表，分配一个变长数组 FT 作为文件顺序表。对文本库中的文件顺序编号，按编号处理文件。

2)选定一个文件，在文件顺表中记录路径，并读取文件的内容，记录尾字符对应逻辑上文本串 T 的位置。对文本内容进行连续的 q-gram 拆分(包括上一个文件中剩余的 $q-1$ 个

字符),并记录其在文本串 T 中对应的偏移量。采用式(2)、式(3)计算每个 q-gram 项的哈希值定位其在词汇表中的位置,同时把每个 q-gram 对应文本串 T 的位置添加到相应的倒排列表中。选定下一个文件,转 2)。

3)直到文本库中所有的文件都处理完毕,q-gram 索引建立完成。

在索引创建过程中,文件按顺序编号依次处理,且每个新地址都添加到倒排列表的尾部,因此 q-gram 索引中所有倒排列表都是按地址升序顺序排列的。如图 1 中的基因库,采用式(1)作为映射函数,当 $q=2$ 时建立的 q-gram 索引如图 2 所示。

3 任意连续种子地址集合的计算

3.1 种子及地址集合计算理论分析

q-gram 索引中存储了文本串 T 中所有长度为 q 的子串的地址信息,通过 q-gram 索引中的信息能够计算出任意长度连续种子在文本串 T 中出现的地址集合。根据连续种子的长度值 m 不同,相应的计算理论和方法也不同,以下分 3 种情况进行讨论:即 $m=q$、$m<q$ 和 $m>q$。

3.1.1 连续种子长度值 $m=q$

当连续种子的长度 m 正好等于索引项长度 q 时,对于给定的连续种子 S 只需通过式(2)计算其哈希值($n=q$),即哈希表中的位置。该哈希地址后的倒排列表就是种子 S 在文本串 T 中出现的地址集合。

3.1.2 连续种子长度值 $m<q$

当连续种子的长度 m 小于索引项长度 q 时,该种子的地址分布在多个索引项的倒排列表中,如种子 GTG 的地址可在 AGTG,CGTG,…等索引项的倒排列表中。另外,如文本存在形如"…CGTGA…"的串,设 GTG 的地址为 p,则在 CGTG 的倒排列表中存储的地址为 $p-1$,而在 GTGA 的倒排列表中存储的地址为 p。因此,计算种子 S 的地址集合并不需要所有包含种子的索引项都参加运算。首先需要确定哪些索引项参加运算,即确定哈希地址集合。对于给定的种子 S,可以通过固定种子位置的方法来确定哈希地址集合,而不必关心其他位置的种子(否则会得到重复的地址)。然后处理哈希地址集合中每个哈希地址后的倒排列表。最后再补充上索引中遗失的地址就得到了种子 S 的地址集合。固定种子位置包括前置种子、中置种子和后置种子,中置种子法与前置、后置法类似这里不讨论。

1. 前置种子法

该方法首先用种子构造一个长度为 q 的字符串 B,指定该字符串中前连续 m 个字符为种子,而后 $q-m$ 个字符为任意字符。如此构造的字符串的哈希值计算可由式(2)变形为式(4)。

$$H(B)=H(S)+\sum_{j=m}^{q-1}x_j\sigma^j \tag{4}$$

其中,$H(S)=\sum_{i=0}^{m-1}I(S[i])\sigma^i$,而因后 $q-m$ 个字符为任意字符,所以 $x_j(m\leqslant j\leqslant q-1)$ 可取$[0,\sigma-1]$中任意值。通过分析式(4)可知,对于给定的种子 S,其哈希地址前部分为固定值,而后部分的可能取值为一个等差数列,最小值为 0,公差为 σ^m,且最大值为$(\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j)\times\sigma^m$,即 $H(S)$ 为首个哈希地址,以后依次加 σ^m 为下一个哈希地址,即哈希地址序列为:

$$H(S),H(S)+\sigma^m,H(S)+2\times\sigma^m,\cdots,H(S)+(\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j)\times\sigma^m \tag{5}$$

最后,合并这些哈希地址后的倒排列表就得到了地址集合 A。此时集合 A 中尚缺少逻辑文本串 T 中最后一个 q-gram的后 $q-1$ 个字符中包含的种子的地址。因此建立索引时需要在字符数组 C 中存储文本库中最后 $q-1$ 个字符及其地址,通过扫描数组 C 计算其中存在的种子及地址,并添加到集合 A 中,此时修正后的地址集合 A 就是种子 S 在文本串 T 中出现的地址集合。

2. 后置种子法

该方法中构造的长度为 q 的字符串 B 中,指定该字符串中后连续 m 个字符为种子,而前 $q-m$ 个字符为任意字符。如此构造的字符串的哈希值计算可由式(2)变形为式(6)。

$$H(B)=\sum_{j=0}^{q-m-1}x_j\sigma^j+H(S)\sigma^{q-m} \tag{6}$$

其中,$H(S)$同上,$x_j(0\leqslant j\leqslant q-m-1)$可取$[0,\sigma-1]$中任意值。因此对于给定的种子 S,后部分为定值,而前部分的可能取值为一个等差数列,最小值为 0,公差为 1,且最大值为$\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j$,即入口范围为$[H(S)\sigma^{q-m},\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j+H(S)\sigma^{q-m}]$,如图 3 中种子 GTG 使用后置种子法得到的哈希地址范围为[184,187]。

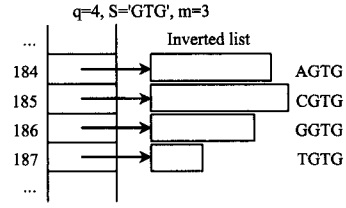

图 3 后置种子法实例

最后,把这些哈希地址后的倒排列表中的每个地址分别加上 $q-m$ 后得到种子 S 的真正地址,然后把恢复的地址添加到集合 A 中,此时集合 A 中尚缺少逻辑文本串 T 中前$q-1$ 个字符中包含的种子地址。因此索引建立时要在字符数组 C 中存储文本库中前 $q-1$ 个字符及其地址,通过扫描数组 C 计算其中存在的种子及地址,并添加到集合 A 中,此时修正后的地址集合 A 就是种子 S 在文本中出现的地址集合。

前置种子法和后置种子法各有优点:前置种子法不需要地址差值变换运算,运算量少;而后置种子法能得到连续的哈希地址,容易处理。

3.1.3 连续种子长度值 $m>q$

当连续种子的长度 m 大于索引项长度 q 时,因单个索引项长度不足而无法确定种子的地址,因此需要多个索引项共同确定其地址集合,如图 4 中的索引中 $q=3$,种子 $S=$'CGAGGCC'可由 3 个索引项'CGA','GGC'和'GCC'共同确定其地址集合。

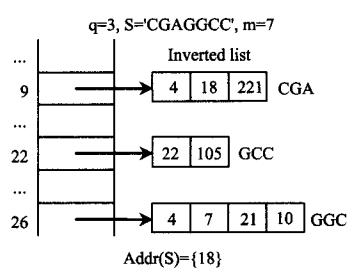

图 4 种子长度大于索引项长度地址计算实例

为计算地址集合，首先对种子 S 进行连续但不重叠的 q-gram拆分，可得到 $w(w\geqslant 1)$ 个完整的 q-gram 项，如最后存在长度不足 q 的剩余串则从后向前重叠取一个 q-gram 项，如图 4 中种子 S 的最后一个 q-gram 为 GCC 与前面的 GGC 重叠 2 个字符。因此最后一个 q-gram 在 S 中的位置为 $m-q$。对于给定种子 S，其地址集合计算步骤如下：

1)置 $j=0$，第一个 q-gram 项为 $S[j,j+q-1]$，采用式(2)计算其哈希地址，并提取该哈希地址后的倒排列表记为集合 A_1。

2)$j=j+q$，此时有 3 种情况：

①如 $j<\lfloor m/q\rfloor\times q$ 时还存在完整的 q-gram，提取q-gram 项 $S[j,j+q-1]$，采用式(2)计算其哈希地址，并提取该哈希地址后的倒排列表记为集合 A_2。对集合 A_1 和集合 A_2 进行差 j 交运算。差 j 交运算规则如下：

$$A_1 \underset{+j}{\cap} A_2=\{t\mid t\subset A_1 \wedge (t+j)\subset A_2\} \tag{7}$$

在式(7)中，其运算的含义为如集合 A_1 中存在地址 t，同时集合 A_2 中存在地址值 $t+j$，则 t 为交运算的结果。最后，用 $A_1 \underset{+j}{\cap} A_2$ 的结果替换 A_1，即集合 A_1 中始终存储的是串 $S[0,j+q-1]$的地址集合，转 2)。

②如 $j==\lfloor m/q\rfloor\times q$ 且 $j<m$ 时存在不足 q 的剩余串，置$j=m-q$，提取最后一个 q-gram 项 $S[j,m-1]$，计算哈希并提取地址列表记为 A_2，计算 $A_1 \underset{+j}{\cap} A_2$ 并替换 A_1，转 3)。

③如 $j==m$ 时刚好无剩余串，转 3)。

3)此时 A_1 为种子 S 的地址集合。

3.2 任意长度连续种子地址集合的计算算法

算法名称：ComputeSeedPositionList

输入：种子 S、q-gram 索引 QI。

输出：种子 S 在逻辑文本串 T 中出现的地址集合 A。

功能：计算给定种子 S 的地址集合，其中 getList(HT[t])函数功能为提取 HT[t]后的倒排列表。

```
1. ComputeSeedPositionList(S,QI)
2. {m=length(S);/*计算种子长度*/
3.   A=∅;B=∅;
4.   if (m==q) /*长度等于 q,直接提取*/
5.   {t=Σ_{i=0}^{q-1} I(S[i])σ^i;A=getList(HT[t]);}
6. else if (m<q) /*长度小于 q,后置种子法*/
7.   {t=H(S)σ^{q-m};v=t+Σ_{j=0}^{q-m-1}(σ-1)σ^j;
8.     for(i=t;i<=v;i++)
9.     {B=getList(HT[i]);A=A∪(B+q-m)}}
10. else /*长度大于 q,连续交运算*/
11. {j=0;t=H(S[j,j+q-1]);A=getList(HT[t]);
12.   for(j=q;j<m-1;j+=q)
13.   {t=H(S[j,j+q-1]);B=getList(HT[t]);
14.     A=A ∩_{+j} B}
15.   j=m-q;t=H(S[j,m-1]);B=getList(HT[t]);
16.   A=A ∩_{+j} B}
17. return A;}
```

4 索引的时空分析

为便于分析，这里设文本库的总长度为 n，字母表大小为 σ，q-gram 索引项长度为 q，种子长度为 m。

4.1 空间需求分析

本文索引的词汇表采用一维数组作为哈希表，占用存储空间为 σ^q。文本库中总 q-gram 数约为 n，因此倒排列表总空间需求为 $O(n)$，因此地址列表的平均长度为 n/σ^q。文件顺序表的大小由文本库中文件数目决定，且一般要远小于 n，可忽略。因此，本文采用的 q-gram 索引总空间需求为 $O(n+\sigma^q)$。

4.2 时间复杂度分析

4.2.1 索引建立时间复杂度分析

建立 q-gram 索引只需扫描一次文本库，同时提取并哈希 q-gram 项再插入地址。用式(2)计算第 1 个 q-gram 的哈希值需要 $O(q)$时间，用式(3)计算下一个 q-gram 项的哈希值只需 $O(1)$时间，因此平均提取并哈希一个 q-gram 的时间是 $O(1)$。所以索引建立的总时间复杂度为 $O(n)$，而不是 $O(nq)$。

4.2.2 计算种子地址时间复杂度分析

索引建立后，在 q-gram 索引中定位一个给定的 q-gram 项的倒排列表的时间需求为 $O(q)$，地址列表的平均长度为 n/σ^q。给定任意一个种子，计算地址集合并访问一遍的时间消耗分析如下：

1)当 $m=q$ 时，只需定位一个 q-gram 项，计算哈希地址时间为 $O(q)$。访问地址列表的时间消耗为 $O(n/\sigma^q)$。因此总时间复杂度为 $O(q+n/\sigma^q)$。

2)当 $m<q$ 时，计算第一个 q-gram 的哈希地址时间消耗为 $O(q)$，从第 2 个哈希地址开始就可以通过公差在 $O(1)$时间内计算得出(不论前置种子还是后置种子)，共 $\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j$ 个，哈希地址计算时间复杂度为 $O(q+\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j)$。总共需要访问 $\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j+1$ 个 q-gram 项的倒排列表，访问地址列表的时间消耗为 $O((\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j+1)\times n/\sigma^q)$。因此总时间复杂度为 $O(q+(\sum_{j=0}^{q-m-1}(\sigma-1)\sigma^j+1)\times(n/\sigma^q+1))$。

3)当 $m>q$ 时，计算种子地址集合的过程需要计算 $\lceil m/q\rceil$ 个 q-gram 项哈希地址，所以计算哈希地址消耗时间为 $O(\lceil m/q\rceil q)$，约为 $O(m)$。因地址列表有序，一次地址列表的差 j 交运算时间消耗约为 $2n/\sigma^q$，计算地址列表时需要进行 $\lceil m/q\rceil-1$ 次交运算，且每次交运算后地址长度缩短，因此地址交运算平均时间消耗不超过 $O(2n/\sigma^q\times(\lceil m/q\rceil-1))$。总时间复杂度为 $O(m+2n/\sigma^q\times(\lceil m/q\rceil-1))$。

5 实验

5.1 实验环境

本文的实验数据来源于美国国家生物技术信息中心NCBI的人类不同完整基因序列(UniGene Build #223，Homo sapiens)[27]，该库中共约 164MB 基因序列文本，共 123252 个完整人类不同基因序列。

为构建文中所提出的 q-gram 索引，实验中选择基因库中 89.6MB 的基因序列作为实验数据(共 82341427 个字符)，为测试在本文索引中进行提取种子地址集合实验，随机生成了长度分别为 6,7,8,9,10 和 11 的 6 个种子集合，其中每个种子集合中含有 1000 个种子。为便于对比索引的空间需求，字符映射函数采用式(1)，哈希表中每个指向倒排列表的指针采

用 4 个字节进行存储，实验中 q-gram 索引的倒排列表中每个地址统一采用 4 个字节进行存储。

文中所有实验都在同一软硬件环境下进行。硬件环境：处理器 AMD X4 630 和主存 4GB；软件环境：Windows XP SP3 和 VISUAL C++ 6.0。

5.2 前置种子法和后置种子法性能对比

当种子的长度小于 q 值时，可以选择前置种子算法和后置种子算法进行种子地址集合的提取。为对比前置种子法和后置种子法的性能差异，实验中取索引项长度 q 值为 11。分别使用长度为 6,7,8,9,10 的种子集合采用两种算法进行种子地址集合提取实验，两种算法随种子长度变化的性能差异统计如图 5 所示。

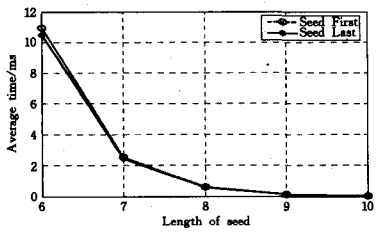

图 5 前置种子法和后置种子法

由图 5 可知，两个算法性能差异很小。后置种子法的性能稍好于前置种子法，这是因为后置种子法在哈希表中使用连续哈希地址计算地址集合所节省的时间要多于其恢复地址操作所消耗的时间。

5.3 多种子时 q 值分析

当进行单种子近似串匹配时，索引项长度 q 的取值等于种子长度时种子地址集合提取速度最快。但当采用多种子时，需要慎重选择 q 值才能使得在索引中提取种子地址集合的平均速度最快。本实验将分别统计采用不同 q 值、多个种子时地址集合的平均提取时间，通过地址平均提取时间来衡量对应 q 值的好坏。

首先进行双种子地址集合提取实验，这里选择种子的长度为 8 和 10，实验中用 8 和 10 种子集合分别采用不同的 q 值进行地址集合提取实验，实验结果统计如图 6 所示。

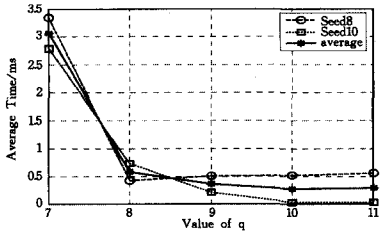

图 6 q 值分析之二种子

由图 6 可知，总体而言，随着 q 值增大，因地址数量减少而使得其提取时间逐渐减少，而当种子长度等于 q 值时地址集合提取的速度最快。由这两个种子的平均获取时间变化曲线可知，当 q 值等于最长种子的长度时平均提取时间最短。这是因为基因数据库的字母表相对较小，使得地址集合合并运算（$m<q$ 时）的速度要快于地址集合交运算（$m>q$ 时）。

然后再进行三种子地址集合提取实验，这里选择了种子的长度分别为 8,9,11，实验中用这 3 个种子集合分别采用不同的 q 值进行地址集合提取实验，实验结果统计如图 7 所示。

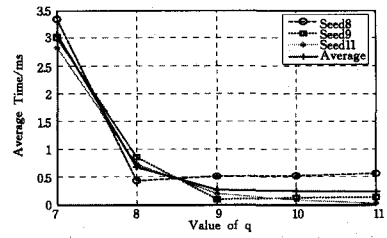

图 7 q 值分析之三种子

由图 7 可知，当 q 值等于最长种子长度时，3 个种子的地址集合平均提取时间最短。

进行近似串匹配时，当使用多个种子时它们的长度一般相差都不大，由以上多种子实验可得出如下结论：

1）q 值越大，因地址集合小而获取速度越快。

2）当文本库字母表较小时，选择 q 值为最长种子长度时地址集合提取速度最快。

5.4 性能分析

文献[19]采用双种子双索引的方式解决了多种子的索引问题，即为每个种子建立一个独立的索引结构，这里称为多索引。此种解决方案的优势是使得每个种子的地址集合提取速度达到最快，但当种子数量增多时多索引的内存空间消耗将成倍增大，最后使得内存空间无法容纳这些索引。虽然使用虚拟内存能弥补内存空间的不足，但这将严重降低地址集合的提取速度。

5.4.1 存储空间消耗对比

本实验中统计了单种子、双种子、三种子时多索引方案和本文索引方案的内存空间消耗情况，对比如表 1 所列。

表 1 多索引和本文索引的存储空间消耗对比

	单种子(任意)	双种子(8,10)	三种子(8,9,11)
本文索引	相同	318MB(10)	330MB(11)
多索引	相同	632MB(8+10)	959MB(8+9+11)

从表 1 中的数据对比可知，当采用多种子时本文的索引只有一个最长种子的索引，空间消耗较小，如三种子时只需存储 q 值为 11 的索引。而多索引的存储空间消耗与种子的数量和长度有关，如三种子时，需要分别存储 q 值为 8,9 和 11 共 3 个索引。多索引的空间消耗随种子数量成倍增长，因此较难用于大型文本库的处理。而在大型文本库的处理中，有限的内存更容易容纳本文索引，因此本文索引在大型文本库处理中具有一定的优势。

5.4.2 索引读取速度对比

实验中还进行了多索引方案和本文索引方案的地址集合平均提取时间对比，对比结果如表 2 所列。

表 2 多索引和本文索引的地址集合提取时间对比

	单种子(任意)	双种子(8,10)	三种子(8,9,11)
本文索引	相同	0.272 ms	0.234 ms
多索引	相同	0.226 ms	0.175 ms

由表 2 的数据对比可知，多种子多索引方案的地址集合提取速度达到最佳，虽然本文索引方案的地址集合提取速度略慢于多索引，但实验数据显示本文索引方案的地址集合提取速度也非常快，且对算法的总体性能影响并不大。尤其在大型文本库处理中，内存中是否能完全容纳索引直接决定了地址集合的提取速度。因此，在多种子近似串匹配中本文索

引方案较适合对大型文本库的处理。

结束语 本文提出了一种能够支持多种子近似串匹配的q-gram索引结构，通过该索引能够快速获取任意长度连续种子的地址集合。文中首先详细介绍了所使用的q-gram索引结构，然后详细地阐述了获取任意长度种子的地址集合的相关理论，最后给出了在本文索引中快速提取任意连续种子地址集合的相关算法。相关实验数据表明，相比文献[19]中的多索引方案，本文的索引方案在牺牲很小的速度的基础上就成倍地减少了索引空间的消耗。因此，本文索引方案较适合作为大型文本库多种子近似串匹配的索引结构。

虽然本文索引方案减少了多种子近似匹配时索引空间的消耗，但索引速度却有所下降，本文将进一步研究索引的优化方法，以提高索引的性能。另外，近似串匹配中留空种子的研究已成为热点问题，作者将研究支持多留空种子的索引方法。

参 考 文 献

[1] Dorneles C F, Goncalves R, Mello R D. Approximate data instance matching: a survey[J]. Knowledge and Information Systems, 2011, 27(1): 1-21

[2] Navarro G. A guided tour to approximate string matching[J]. ACM Computing Surveys, 2001, 33(1): 31-88

[3] Lu C W, Lu C L, Lee R C T. A new filtration method and a hybrid strategy for approximate string matching[J]. Springer Berlin Heidelberg, 2013, 20: 143-155

[4] Levenshtein V. Binary codes capable of correcting deletions, insertions, and reversals[J]. Soviet Physics Doklady, 1966, 10(8): 707-710

[5] Burkhardt S. Filter algorithms for approximate string matching [D]. Saarland: Department of Computer Science, Saarland University, 2002

[6] Tian Y, Tata S, Patel J M, et al. Practical methods for constructing suffix trees[J]. VLDB Journal, 2005, 14(3): 281-99

[7] Manber U, Myers G. Suffix arrays: A new method for on-line string searches[J]. SIAM Journal on Computing, 1993, 22(5): 935-948

[8] Navarro G, Baeza-yates R. A practical q-gram index for text retrieval allowing errors[J]. CLEI Electronic Journal, 1998, 1(2): 1-16

[9] Kim M S, Whang K Y, Lee J G. n-Gram/2L- approximation: A two-level n-gram inverted index structure for approximate string matching[J]. Computer Systems Science and Engineering, 2007, 22(6): 365-379

[10] Puglisi S J, Smyth W F, Turpin A, Inverted files versus suffix arrays for locating patterns in primary memory[C]// Proceedings of the 13th Symposium on String Processing and Information Retrieval. Berlin, Germany: Springer Verlag, 2006: 122-133

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

[12] Altschul S F, Madden T L, Alejandro A S, et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs[J]. Nucleic Acids Research, 1997, 25(17): 3389-3402

[13] Wu S, Manber U. Fast text searching allowing errors[J]. Communications of the ACM, 1992, 35(10): 83-91

[14] Chang Y I, Chen J R, Hsu M T. A hash trie filter method for approximate string matching in genomic databases[J]. Applied Intelligence, 2010, 33(1): 21-38

[15] Burkhardt S, Crauser A, Ferragina P, et al. Q-gram based database searching using a suffix array[C]// Proceedings of the Annual International Conference on Computational Molecular Biology, RECOMB 99. New York, USA: ACM, 1999: 77-83

[16] Jokinen P, Ukkonen E. Two algorithms for approximate string matching in static texts[C]// 16th International Symposium Proceedings on Mathematical Foundations of Computer Science. Berlin, Germany: Springer-Verlag, 1991: 240-248

[17] Rasmussen KR, Stoye J, Myers EW. Efficient q-gram filters for finding all epsilon-matches over a given length[J]. Journal of Computational Biology, 2006, 13(2): 296-308

[18] Sutinen E, Tarhio J. Filtration with q-samples in approximate string matching[C]// Proceedings of 7th Annual Symposium on Combinatorial Pattern Matching, CPM 96. Berlin, Germany: Springer-Verlag, 1996: 50-63

[19] Ma B, Tromp J, Li M, et al. PatternHunter: faster and more sensitive homology search[J]. BIOINFORMATICS, 2002, 18(3): 440-445

[20] Egidi L, Manzini G. Better spaced seeds using Quadratic Residues[J]. Journal of Computer and System Sciences, 2013, 79(7): 1144-1155

[21] Baeza-Yates R, Navarro G. Faster approximate string matching [J]. Algorithmica, 1999, 23(2): 127-158

[22] Myers E W. A sublinear algorithm for approximate keyword searching[J]. Algorithmica, 1994, 12(4/5): 345-374

[23] Ilie S. Efficient computation of spaced seeds[J]. BMC Research Notes, 2012, 5(1): 1-7

[24] Kim Y, Park H, Shim K. Efficient processing of substring match queries with inverted variable-length gram indexes[J]. Information Sciences, 2013, 244: 119-141

[25] Yang X, Wang B, Li C. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2008: 353-364

[26] Karp R M, Rabin M O. Efficient randomized pattern-matching algorithms[J]. IBM Journal of Research and Development, 1987, 31(2): 249-260

[27] NCBI. UniGene Build #223, Homo sapiens[DB/OL]. (2010-1-27)[2010-04-28]. ftp. ncbi. nih. gov//repository/ UniGene/Homo_sapiens/Hs. seq. uniq. gz.

(上接第252页)

[13] Kalman D. A singularly valuable decomposition: the SVD of a matrix[J]. College Math Journal, 1996

[14] Golub G H, Van Loan C F. Matrix computations [M]. Baltimore, MD, USA: Johns Hopkins University Press, 1996: 374-426

[15] 盖杰，王怡，武港山. 潜在语义分析理论及其应用[J]. 计算机应用研究，2004，21(3)：9-12

[16] 宁健，林鸿飞. 基于改进潜在语义分析的跨语言检索 [J]. 中文信息学报，2010，24(3)：105-111

[17] 于江生，俞士汶. 中文概念词典的结构[J]. 中文信息学报，2002，16(4)：12-20

[18] 王卫国，徐炜民. 基于潜在语义分析的个性化查询扩展模型[J]. Computer Engineering，2010，36(21)：43-45